Programación BASIC para microcomputadoras

Luis Joyanes Aguilar



Amstrand - Commodore - MSX - Spectrum
Apple II - HP - IBM/PC y compatibles



2ª EDICION

Programación BASIC para microcomputadoras

SEGUNDA EDICION

CONSULTORES EDITORIALES
AREA DE INFORMATICA Y COMPUTACION

Antonio Vaquero Sánchez
Catedrático de Informática
Facultad de Ciencias Físicas
Universidad Complutense de Madrid
ESPAÑA

María Lourdes Fournier García
Actuaria, Facultad Ciencias UNAM
Profesora Asociada a Tiempo Completo
Universidad Autónoma Metropolitana
MEXICO

Gerardo Quiroz Vieyra
Ingeniero en Comunicaciones y Electrónica
Escuela Superior de Ingeniería Mecánica y Eléctrica IPN
Carter Wallace, S. A.
Universidad Autónoma Metropolitana
Docente DCSA
MEXICO

Alfonso Pérez Gama
Ingeniero Electrónico
Universidad Nacional de Colombia
COLOMBIA

José Portillo
Universidad de Lima
PERU

Luis Ernesto Ramírez
Coordinador de Informática
Escuela de Administración y Contaduría
Universidad Católica Andrés Bello, UCAB
VENEZUELA

Programación BASIC para microcomputadoras

SEGUNDA EDICION

Luis Joyanes Aguilar
Jefe del Area de Lenguajes de Programación
Cursos de Informática
Facultad de Sociología León XIII. Madrid
Universidad Pontificia de Salamanca

McGraw-Hill
MADRID • BOGOTÁ • BUENOS AIRES • GUATEMALA • LISBOA • MÉXICO • NUEVA YORK
PANAMÁ • SAN JUAN • SANTIAGO • SÃO PAULO
AUCKLAND • HAMBURGO • LONDRES • MONTREAL • NUEVA DELHI • PARÍS
SAN FRANCISCO • SINGAPUR • ST. LOUIS • SIDNEY • TOKIO • TORONTO

Obra declarada de utilidad para el Ejército del Aire (B. O. D. núm. 81, Orden 522/14694/año 1985).

Obra declarada de utilidad para el Ejército de Tierra (B. O. D. núm. 99, Orden 313/16725/año 1985).

PROGRAMACION BASIC PARA MICROCOMPUTADORAS 2/E.

Atlacomulco 499-501, Naucalpan de Juárez, Edo. de México
Miembro de la Cámara Nacional de la Industria Editorial, Reg. núm. 465

ISBN: 968-422-019-7

ISBN: 84-7615-127-6
Depósito legal: M. 16345-1988

Compuesto en FER, Fotocomposición, S. A. Lenguas, 8 - 28021 Madrid.
Impreso en Artes Gráficas EMA, S. A. Miguel Yuste, 27 - 28037 Madrid.

De esta edición se imprimeron 6.000 ejemplares en mayo de 1988.

PRINTED IN SPAIN - IMPRESO EN ESPAÑA

A mi esposa, Gracia, cuya comprensión y cariño nunca me faltan, y a mi madre, que si aún viviera, este libro la habría hecho inmensamente feliz.

CONTENIDO

PROLOGO

A LA PRIMERA EDICION

Es para mí una doble satisfacción que el autor me haya dado la oportunidad de prologar esta obra de programación en lenguaje BASIC; en primer lugar porque veo en ella el trabajo minucioso y perfectamente planeado de un antiguo alumno de la Academia de Artillería, el Capitán Joyanes, de cuya formación, aunque sea en una proporción muy modesta, me considero partícipe, y en segundo lugar porque la obra, fruto de la experiencia acumulada por el autor en su labor de profesorado, en la actual Academia de Artillería, la considero una nueva aportación técnica del Arma a la comunidad civil.

Es de sobra conocido que las primeras computadoras nacieron, entre otros fines, para satisfacer la necesidad de resolver con rapidez los laboriosos cálculos necesarios para determinar los distintos parámetros de la trayectoria de los proyectiles de Artillería, que después habrían de plasmarse en las tablas de tiro. Desde entonces las computadoras acompañan al artillero en su trabajo diario, ya que forman parte de los Sistemas de Dirección de Tiro de la Artillería de Campaña, Costa y Antiaérea, del planeamiento y dirección automática de los fuegos y de los sistemas de simulación, que son básicos en la instrucción moderna y en la valoración de los métodos de combate. A nadie debe extrañar por lo tanto que la Academia de Artillería dedique una atención especial a los temas informáticos y que por limitaciones de tiempo el profesorado se vea obligado a una labor de concreción y síntesis, de la que es una elocuente muestra este interesantísimo trabajo del autor.

José Pontijas de Diego
General de División
Jefe de Artillería del Ejército

Por unas u otras razones, el lenguaje de programación de alto nivel más popular entre el gran público es, sin duda alguna, BASIC. Por ello se necesitan textos que cubran esta área. Hay muchos libros disponibles en inglés dirigidos a toda clase de usuarios, incluidos los absolutamente debutantes. Existen también traducciones a nuestra lengua de algunos de ellos y, aunque no abundan las buenas traducciones, las hay excelentes. Pero creemos que es bueno que aparezcan obras directamente en español para el público de habla hispana. Esta práctica es poco usual entre los profesores y especialistas técnicos de nuestra habla. Sin embargo, creemos que es posible y deseable que se frecuente la producción de textos técnicos por nuestros potenciales autores. Es posible porque aunque nuestro nivel científico y tecnológico no es tan elevado como debiera, existen profesionales con suficiente solvencia científica y experiencia pedagógica para cubrir este hueco. Este libro es una buena muestra de ello. Por otra parte es deseable la aparición de obras técnicas propias para reducir la colonización tecnológica en el ámbito editorial a cotas soportables.

Esta obra aparece en un momento en que la informática va penetrando en el ámbito del gran público en nuestro país, y en países de nuestra misma lengua al otro lado del Océano. Con un cierto retraso, ciertamente, respecto a los países más avanzados. Pero nunca es tarde. Deseamos que contribuya a la elevación cultural tecnológica de nuestras gentes. Así sea.

Antonio Vaquero Sánchez
Catedrático de Informática
Facultad de Ciencias Físicas
Universidad Complutense de Madrid

PROLOGO

A LA SEGUNDA EDICION

INTRODUCCION

La primera edición —marzo 1984— la comenzábamos preguntándonos *¿por qué un libro más de BASIC?* si existían tantos en el mercado. En estos dos años y medio transcurridos desde la aparición de la primera edición, la pregunta sigue teniendo vigencia, entre otras cosas porque, por una parte BASIC sigue siendo uno de los lenguajes más utilizados universalmente y además todas las computadoras domésticas y personales, junto con las microcomputadoras, se suelen comercializar con una versión de BASIC incluida en el software del equipo, y, por otra parte, esta obra quiere significar una modesta aportación a la ya importante —por suerte— bibliografía. En la primera edición contestábamos arguyendo que el mundo informático estaba cambiando y que las *computadoras personales* ya no eran uso exclusivo de una pequeña élite, sino que cualquier persona, estudiante, profesional, ingeniero, puede acceder a ese sugestivo mundo, y, en consecuencia, se necesitaban obras escritas para sus niveles que les permitiera un rápido aprendizaje, y que por ello el problema no era que hubiese muchas obras de BASIC sino que todas y cada una de ellas aportaran su grano de arena a la *alfabetización informática* (término acuñado por los sajones como «computer literate» y por los francófonos como «deuxième alphabetization»). Ese es nuestro objetivo, aportar nuestro grano de arena, con un libro estructurado.

Hoy día ya se venden pocos micros domésticos o familiares, con memorias por debajo de 64K, e incluso existen modelos de *128K y unidad de disco* o casete integrado. Sin embargo aunque éste es el panorama en la microinformática doméstica, donde realmente se ha sentido el empuje del progreso ha sido en la *microinformática profesional:* IBM con sus modelos PC, XT y AT, junto con la saga de compatibles (Olivetti, Toshiba, Ericcson, Computec, Bull, HP, etc.) dominan el mercado. Apple Macintosh, con su reciente modelo Plus y las características especiales que supone su sistema operativo y el ratón, trata de luchar contra el *gigante azul* (IBM), pero a pesar de sus excelencias, el fenómeno PC es quien se lleva el gato al agua. Sin embargo, la filosofía Macintosh ha prendido en el campo informático y ya son muchos sus seguidores: Atari 520ST y 1040ST, AMIGA de Commodore, etc.

Las memorias centrales en los micros profesionales han superado la barrera de los 256K y en 1986 los 640K son ya casi imprescindibles para trabajo profesional, y comienzan a aparecer los micros con memoria central de 1M en una sola tarjeta, como es el caso del Macintosh Plus, IBM AT, etc. —con aplicaciones los PC,s llegan a 8M y 16M.

Se puede considerar, y así lo hacen los analistas informáticos, que el micro doméstico, con sus actuales prestaciones, y el micro profesional están pasando a ser un bien de consumo familiar y empresarial.

Esta obra pretende ser útil tanto para personas informáticas como no informáticas, que decidan aprender a programar en lenguaje BASIC, y trata de constituir el primer paso para la formación en este lenguaje. Para ello hemos utilizado un lenguaje interactivo —como son los de los ordenadores o computadoras personales— para comunicarnos con el lector y hemos recurrido a estudios comparativos en la medida en que el espacio físico lo permitía, con modelos de prestigio y populares: IBM PC, MSX, Commodore, Apple, HP, Amstrad, Spectrum, etc.

Cuando salió la primera edición al mercado —primavera de 1984— el *boom* informático en equipos lo constituían: ZX Spectrum, Apple II, el viejo VIC-20 de Commodore, Dragón 32, y comenzaba el reinado del IBM PC; en hardware, los micros de 8 bits (6502 y Z-80) y las memorias de 32-64K eran los significativos, y, en micros de 16 bits (8086 y 8088), el IBM PC con memorias de 64K a 256K marcaban la pauta. Actualmente, verano de 1986, las

cosas han cambiado a velocidades vertiginosas; el mítico Spectrum se sigue comercializando, pero su influencia y prestigio han cesado; las nuevas generaciones de 8 bits tienen la base de 128K en memoria central (microprocesador Z-80 y en menor medida el 6502); se ha resucitado el viejo sistema operativo CP/M —merced, sobre todo al prestigioso Amstrad— la gama profesional IBM PC y compatibles constituyen, sin duda, el modelo estándar.

Sin embargo, y pese a los espectaculares avances, existe una cosa común en los viejos y en los nuevos modelos, el *intérprete BASIC* (y ya con mucha frecuencia, el *compilador* BASIC) residente o no. Por otra parte, se han fortalecido o han aparecido nuevas y numerosas versiones de BASIC potentes y rápidas: MBASIC, BASICA, MSX, GW BASIC, Amstrad, True BASIC, Better BASIC, Quick BASIC, HP. El lenguaje BASIC está vivo y, pese a sus detractores, se seguirán formando numerosos estudiantes, profesionales, ingenieros, etc., en este lenguaje por sus grandes cualidades: sencillo y potente a la vez, interactivo, versiones para todos los gustos (gestión, científicos, gráficos...), etc.

El avance indudable del lenguaje Pascal como disciplina universitaria, no ha hecho desaparecer los lenguajes clásicos COBOL, BASIC y FORTRAN, y durante mucho tiempo convivirán, entre otras cosas por su complementariedad y la facilidad que supone para el aprendizaje de un lenguaje de programación el conocimiento de otro lenguaje. Tal vez la nueva generación de lenguajes C, Ada, ... obligará a replantear los estudios clásicos de lenguajes: por ello, BASIC se está preparando y la aparición de las nuevas versiones potentes y estructuradas, como Quick, True, Better, etc., parecen pretender competir con los lenguajes estructurados tipo Pascal.

CONTENIDO

Aunque las computadoras parecen misteriosas y mágicas, son realmente sólo máquinas que reúnen y ordenan información. Las computadoras no tienen inteligencia, aunque a veces pueda parecerlo. Pensemos en las computadoras que conocemos de las películas de «La Guerra de las Galaxias, R2D2 y C3PO; son ejemplos de computadoras en forma de robots que tienen personalidad «humana» y que han sido programados para desempeñar esta misión. ¿Es posible que una persona no profesional de la informática pueda programar esas máquinas? La contestación evidentemente no puede ser tajante, y, sin embargo, el fenómeno de las computadoras personales ha sido posible no sólo por la miniaturización de los componentes electrónicos («chips» o «circuitos integrados»), sino por la evolución y desarrollo de lenguajes de alto nivel populares y de fácil aprendizaje. El lenguaje BASIC ha sido uno de ellos y hoy día desde los niños de enseñanza primaria a los universitarios, o desde las amas de casa a los ingenieros, pueden desarrollar sus programas en dicho lenguaje para realizar juegos de vídeo, cálculos matemáticos, el presupuesto familiar o controlar pequeños electrodomésticos caseros.

Esta obra se dirige a la vez al lector principiante en informática, y al lector que teniendo ya experiencia en otros lenguajes de programación desee conocer de un modo rápido las características esenciales del BASIC.

Para el primer tipo de lectores no se requiere ningún conocimiento previo, y sólo recomendarles que lean detenidamente cada capítulo antes de pasar al siguiente a fin de comprender los conocimientos básicos que necesitarán a medida que avancen en materia. Cada capítulo contiene numerosos ejercicios y programas, incluyendo una sección específica de programas seleccionados y resueltos como aplicación de los conceptos aprendidos en él. Trate de realizar los programas en un cuaderno o en la computadora y luego compruebe con las soluciones del libro. Asimismo incluye dos secciones más con preguntas y programas de repaso con el objeto de que el lector pueda comprobar su estado de conocimientos.

Para los lectores ya iniciados en informática, el método de estudio debe ser diferente; pueden pasar directamente al capítulo 2 e iniciar el estudio del lenguaje ensayando directamente todos los ejercicios propuestos, recomendándoles una rápida lectura del capítulo 1 para familiarizarse con las peculiaridades específicas del BASIC, si es que no las conocen ya.

El último capítulo les proporciona unas normas para la puesta a punto o depuración de los programas, lo que les permitirá asentar todas las ideas adquiridas a lo largo del libro, y

sobre todo un punto de partida para iniciar ya en profundidad la programación real de sus propias necesidades.

Los apéndices proporcionan información necesaria para complementar su introducción en el *maravilloso* mundo de la microinformática y extraer todo el rendimiento posible a su computadora. Se incluyen *códigos ASCII, guías de referencia rápida, características de las computadoras personales más populares del mercado, subrutinas de utilidad y tablas de conversión de sistemas de numeración* que completan estos apéndices.

El capítulo 0 está dirigido a los debutantes en la microinformática y expone de modo sencillo los conceptos y términos usuales, como *computadora, impresora, unidad de disco,* o los más recientes, como ratón, palancas de mando, etc.

El capítulo 1 le introduce en la programación, enseñándole a manejar la computadora como una calculadora (caso no frecuente) o en modo programa (procedimiento usual), enseñándole a escribir o introducir un programa en memoria, cargar y conservar programas en memoria externa (cinta o disco), etc.

El capítulo 2 le proporcionará el *abecé* del BASIC, los elementos del lenguaje de programación: alfabeto, sentencias, órdenes..., así como el tipo de datos y su tratamiento, que podrá manejar en un programa.

El capítulo 3 enseña a utilizar las sentencias conocidas como *elementales:* LET, INPUT y PRINT, con las que usted ya podrá realizar pequeños programas.

El capítulo 4 es de gran importancia, pues aprenderá a controlar sus programas con la realización de bucles y bifurcaciones. Al final de su estudio usted estará en condiciones de realizar programas más complejos. Como dijo Dijkstra (el padre de la programación estructurada), la sentencia GOTO es perjudicial en la programación; aunque didácticamente haremos uso de ella, le recomendamos no abuse de la misma, y si le es posible recurra a sentencias más estructuradas, como IF-THEN-ELSE, WHILE, ON GOSUB, etc., que a buen seguro tendrá su computadora.

El capítulo 5 explica el tratamiento de datos alfanuméricos (letras y números), con lo que conocerá un método para dialogar mejor con su computadora y otras posibilidades.

El capítulo 6 está dedicado esencialmente a los lectores que trabajan en el campo científico, permitiéndoles utilizar funciones aritméticas, logarítmicas, trigonométricas, etc., con lo que dispondrá de una potente herramienta para solucionar sus problemas de matemáticas, física, ingeniería, arquitectura, etc. De cualquier forma, su aplicación es muy amplia, por lo cual le aconsejamos una lectura para que conozca esas posibilidades, aunque no se dedique usted al campo científico o a la ingeniería.

Los capítulos 7 y 8 están dedicados a la entrada y salida de los datos que manejará en el programa, presentación de éstos en pantalla e impresora y formatos adecuados, explicando el uso de las variantes de la sentencia PRINT, SPC, TAB, POS y el manejo de la potente sentencia PRINT USING, usual hoy día en cualquier computadora de tipo profesional y de gestión. Estos capítulos le introducen en el concepto elemental de fichero, y de listas y tablas (*matrices* o *«arrays»*), por lo que su conocimiento será esencial para su formación profesional.

El capítulo 9 le enseñará qué es una subrutina y cómo aumentar la potencia de sus programas con su adecuado uso.

El capítulo 10 está dirigido a funciones y sentencias especiales, que si bien no están en todos los modelos de computadoras, sí son esenciales; su uso facilitará mucho la programación. Las funciones de entrada y salida de datos INP, OUT... le permitirá controlar circuitos electrónicos si usted es aficionado o profesional de la electrónica. Las sentencias CALL, USR... le introducirán en el código máquina; otras sentencias especiales le proporcionarán un reloj a su computador, etc.

Aunque todo el libro ha sido revisado, ampliado y actualizado, es sin duda el capítulo 11 la novedad más querida para mí. El tratamiento o procesamiento de ficheros en disco constituyen este capítulo; sin duda una ausencia notable en la primera edición —razones de tiempo y espacio lo impidieron—. Las cariñosas críticas de mis alumnos, compañeros, lectores, profesores de los distintos niveles de educación, me exigían este esfuerzo complementario. La teoría y práctica de ficheros se ha realizado sobre Microsoft BASIC (la versión estándar más difundida: BASICA del IBM PC, GW BASIC de compatibles, MSX-BASIC, MBASIC, etc.), pero se han dedicado varias páginas al estudio comparativo de otras versio-

nes BASIC, como Applesoft de Apple II, HP, Commodore, Amstrad, etc. Espero dar respuesta a esas cariñosas críticas con este capítulo introductorio a los ficheros, por otra parte de tanta importancia, dejando para otra ocasión una profundización en los mismos.

Por último, el capítulo 12 le enseñará a poner a punto programas proporcionándole normas y reglas para una buena programación, lo que le ahorrará tiempo y trabajo.

Los apéndices, como ya se ha mencionado, son fuentes de información que sólo tratan libros específicos y especializados o revistas técnicas, y que hemos creído necesario incluir en la obra para facilitar la tarea al lector y evitar enojosas *búsquedas* que no siempre acaban con éxito.

AGRADECIMIENTOS

Esta segunda edición, aunque recoge todas mis experiencias didácticas en los dos largos años transcurridos desde la aparición de la primera edición en la primavera de 1984, no hubiera sido posible con el formato y presentación actual, sin la inestimable ayuda que han supuesto para mí las ideas, sugerencias y aportaciones de compañeros, alumnos y lectores, que me han permitido mejorar la obra notablemente. Destacar a tantos como me han dado grandes o pequeñas ideas, críticas constructivas e incluso en ocasiones —por suerte escasas— críticas destructivas, corregido o revisado texto y programas, sería imposible; *a todos, compañeros y alumnos, mi agradecimiento.*

A todos los profesores, alumnos y lectores que me han distinguido, eligiendo este libro para su formación o autoformación.

Sin embargo, no quiero dejar de citar una serie de personas que han influido considerablemente en la materialidad de la obra, y que por suerte para mí me honran con su amistad.

A Ricardo Sotomayor Sáez, antiguo alumno mío en los cursos de Especialización en Electrónica (diploma SDT y DIO) de la Academia de Artillería del Ejército en Madrid —hoy brillante profesor de la Academia de Artillería en Segovia— que revisó y corrigió el primer manuscrito de la primera edición.

A Luis Florit Sánchez, profesor de la Academia de Artillería y mi Jefe —y sin embargo amigo y colaborador en mis obras—, en el Centro de Cálculo y Simulación Balística de la misma, que siempre me alienta en mis trabajos informáticos y sobre todo revisó, detectó y corrigió errores en dos reimpresiones de la primera edición. A Mariano Artal Rozas, antiguo y brillante alumno mío de la Academia General Militar en Zaragoza —hoy comparte conmigo la docencia— que también revisó otra reimpresión.

A profesores y alumnos de los Cursos de Informática de la Facultad de Sociología León XIII de la Universidad Pontificia de Salamanca, por sus sugerencias y críticas constructivas de la primera edición.

Al editor y gerente de la División Universitaria de McGraw-Hill España, Antonio García Maroto, excelente y querido amigo, que tanto cariño pone en mis obras, por sus consejos, sugerencias e ideas para conseguir elevar el nivel del libro, y sobre todo por su enorme paciencia ante mis continuas modificaciones, revisiones, correcciones inserciones, etc., en ese afán *perfeccionista* que nos suele invadir a los informáticos en todos nuestros proyectos.

A Eduardo Jiménez Ferry, director general de McGraw-Hill España, por su amistad y consejos, sus continuas atenciones y por la confianza que siempre deposita en mis obras.

No quiero dejar de citar a Fernando Serrano —gerente de ventas profesionales— de McGraw-Hill España por el cariño profesional con que trata mis obras y al resto del equipo de ventas de McGraw-Hill, Juan Carlos Cavin, Mariano Norte, Eduardo Susanna, etc.

A la empresa STI (SA Tradec Internacional) distribuidora de impresoras Epson por la información técnica facilitada; a Microelectrónica y Control —distribuidora de las computadoras Commodore—, y a Investrónica —distribuidoras de computadoras Sinclair— por sus fotografías; y a McGraw-Hill España por su autorización para reproducir algunas fotografías de su fondo editorial.

Finalmente, deseo expresar mi reconocimiento a mi buen colaborador y gran amigo Alfonso Sanz Alonso —excelente programador donde los haya— que ha listado y depurado con infinita paciencia todos los programas de este libro. Cuando este libro salga a la luz, se

encontrará estudiando Informática en la Universidad Alemana de Fura; mi cariño y mi recuerdo, para ese futuro gran informático.

Sólo queda esperar que el lector de esta obra encuentre en ella las soluciones adecuadas a los problemas que se le plantearán en su formación en programación BASIC y que este libro sea una herramienta válida de estudio y consulta.

Luis Joyanes
Madrid, agosto 1986

CAPITULO 0

Las computadoras y su mundo

0. MICROHISTORIA

La idea de repetir muchas operaciones sencillas para completar grandes proyectos no es nueva. Los faraones del antiguo Egipto utilizaron este concepto para construir las pirámides; con cada uno de los muchos esclavos movían bloques una pequeña distancia centenares de veces en sucesión. Las figuras pueden ser dibujadas repitiendo pequeños puntos de colores diferentes. Este concepto de reducir laboriosas tareas a una serie de tareas repetitivas sencillas es la idea fundamental sobre la computadora.

A mediados del siglo XVII el filósofo, matemático y teólogo francés Pascal tuvo la idea de la primera calculadora mecánica, para lo cual utilizó una serie de engranajes o ruedas dentadas que le permitían realizar sumas y restas. Leibnitz, el famoso científico alemán, construyó otro modelo en 1694 con el que pudo multiplicar y dividir, sumando números repetidamente, pero esta máquina no era mecánicamente rápida ni segura. En Inglaterra, en 1835, Charles Babbage construyó una máquina de realizar cálculos que mejoró con un ambicioso plan de máquina analítica, que si bien no tuvo éxito completo, constituye el primer paso serio en la historia de las computadoras, creando una gran conmoción en el mundo científico; Ada Byron, a quien se conoce a veces como el primer programador, trabajó con la máquina y organizó el esquema lógico de la misma.

El primer americano que construyó una computadora fue el doctor Hans Hollerith, que ante la necesidad de mecanizar el censo de los Estados Unidos de 1890, en menos de cuatro años, diseñó una máquina que leía tarjetas perforadas similares a las diseñadas por Jacquard y Babbage, en el sentido de representar la contestación afirmativa por una perforación y la contestación negativa sin perforación. En esencia, nació la codificación digital (SI, NO, «0», «1») como soporte de información. Hollerith creó una compañía basada en esta nueva máquina, que en 1924 formó parte de la famosa compañía IBM.

La siguiente computadora fue desarrollada en el Instituto de Tecnología de Massachussetts por Vannavar Busch. Utilizó engranajes mecánicos y dispositivos de rotación para representar funciones matemáticas. Estas máquinas conocidas por *computadoras analógicas* se vendieron bien pese a sus limitaciones en velocidad y precisión. En la década de los treinta se adoptó la filosofía digital, utilizando simples interruptores o contactos en estado de activación o desactivación, encendido o apagado, para funcionar con la máquina, de modo que toda la información que se envía a las máquinas se expresa utilizando interruptores múltiples que la máquina lee como *activado* o *desactivado.*

En 1939, IBM construyó la primera gran *computadora digital,* MARK I. Además de muy cara era extremadamente ruidosa. Utilizaba centenares de interruptores electromecánicos. No fue IBM, sin embargo, la primera firma que utilizó sólo interruptores eléctricos. La Universidad de Pensilvania construyó la computadora ENIAC, empleando tubos de vacío. Al igual que MARK I, esta máquina fue grande y extremadamente cara, y en lugar de ser ruidosa tenía el problema del calentamiento; además cada vez que se ponía en marcha, un ejército de ingenieros

y científicos tenía que revisar la máquina cada vez que se utilizaba, ya que cada problema exigía una configuración distinta.

En 1946, J. von Neumann enunció los principios de funcionamiento de una computadora de modo que no fuese preciso modificar los circuitos internos para cada programa y que éste se almacenara en la memoria.

La primera computadora comercial, llamada UNIVAC I, nació en 1951 y se entregó a la Oficina del Censo USA. Esta máquina podía ejecutar centenares de operaciones matemáticas cada segundo y fue considerada extremadamente rápida y eficiente. Esta y las que le siguieron se pueden considerar la *primera generación de computadoras,* utilizaban válvulas de vacío y podían ejecutar unas mil instrucciones por segundo, siendo su campo de aplicación exclusivamente científico y militar.

A comienzos de la década de los sesenta, el advenimiento del transistor, que sustituyó a las válvulas de vacío, creó la *segunda generación de computadoras.* En 1965, la técnica evolucionó y aparecieron los primeros circuitos integrados que reunían en un *chip* o *cápsula miniatura* numerosos transistores, y que ocupan un espacio físico sensiblemente más pequeño. La serie 370 de IBM es un claro ejemplo de estas máquinas, que conforman la *tercera generación de computadoras.*

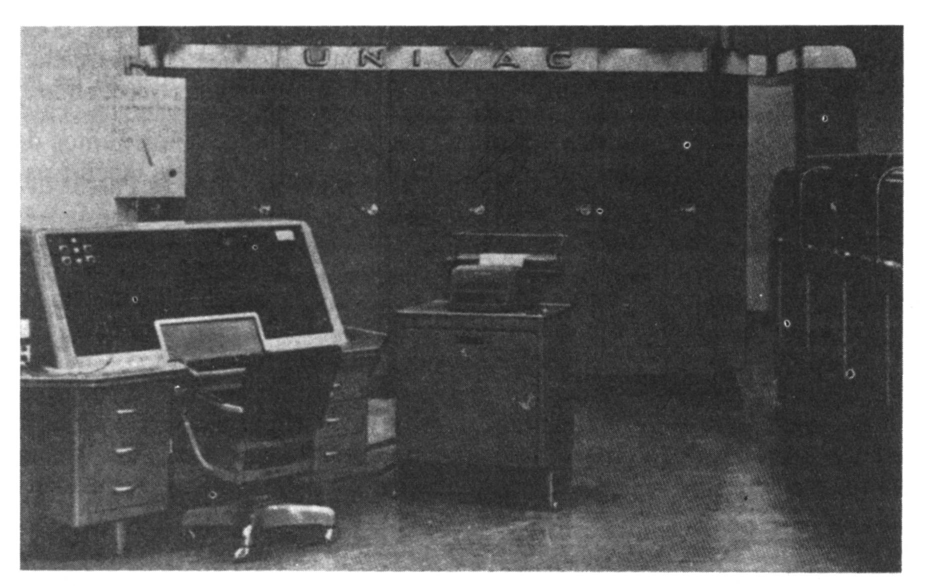


Figura 0.1. UNIVAC (primera computadora disponible comercialmente).

Las técnicas de integración alcanzaron tal desarrollo que al comienzo del año 1970 nació el primer *microprocesador,* que consistía en realidad en la unidad central de proceso de una computadora y el comienzo de la miniaturización de los equipos, así como la creación de *terminales inteligentes* que eran fáciles de construir y hacían las tareas de otras computadoras más grandes y caras. En 1973, una compañía llamada INTEL desarrolló el primer chip de este tipo que se llamó 8008. A partir de esta fecha comenzaron a surgir computadoras basadas en el 8008, y posteriormente en el 8080, que fue el primer microprocesador para usos domésticos. Con este microprocesador nació la revolución de los *micros.* Numerosas compañías se lanzaron al final de la década a la conquista del nuevo mercado: *microcomputadoras.* Apple, Radio Shack y Commodore (todas, firmas americanas) fueron las pioneras y aún hoy día siguen en punta.

En 1977, Apple saca al mercado la ya famosa computadora Apple II, basada en el microprocesador 6502 de Rockwell, que se vendió extremadamente bien y que hoy día con los modelos Apple IIe y Apple IIc sigue siendo una de las firmas más populares del mercado. Durante el mismo año, Radio Shack, filial de Tandy Corporation, lanzó su primer modelo TRS-80 y comenzó a venderla en su extensa cadena de almacenes; éste fue el primer gran punto de venta, el TRS-80 estaba disponible en el mercado a un precio de 600 dólares (unas 40.000 o 50.000 pesetas). También en 1977 Commodore introdujo su primera microcomputadora, PET, que fue la primera computadora integrada en una sola unidad (pantalla, teclado y unidad de casete).

A partir de estas fechas, nuevos y numerosos fabricantes han aparecido en el mercado, y la gama de modelos ofertados es tan amplia que se puede afirmar sin temor a equivocarse que existen microcomputadoras para todos los «gustos» y «bolsillos», desde la ATARI para juegos hasta las computadoras personales IBM, PC, XT y AT, pasando por ZX Spectrum, Amstrad y MSX para pequeña gestión y juegos.

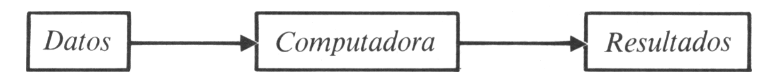
La tecnología de la década de los ochenta ha introducido las potentes máquinas de 16 bits y los periféricos más versátiles: *discos flexibles, discos rígidos, lápices ópticos, digitalizadores, ratones, etc.*

Los modelos Macintosh de Apple con su «ratón», HP 150 de Hewlett Packard con su «pantalla sensible» al tacto, y Amstrad PCW8256 con «su procesador de textos», en un equipo completo (monitor, teclado e imprenta) por menos de 150.000 ptas., son sólo una muestra de la evolución del mercado microinformático en la segunda mitad de la década de los ochenta. Sin embargo, es IBM, con sus modelos XT y AT y su saga de compatibles quien domina el mercado.

1. QUE ES UNA COMPUTADORA

Se puede responder a esta pregunta presentando a la computadora como una máquina electrónica que permite el tratamiento automático de la información. Una computadora se diseña o configura para satisfacer las necesidades de cada usuario o grupo de usuarios.

Para que la computadora pueda tratar automáticamente un problema, se le proporcionará un plan de trabajo preciso: *un programa,* que está constituido por un conjunto de instrucciones a la computadora. El objeto y el interés de la programación es permitir especificar a una computadora un cierto trabajo a efectuar de un modo automático. Para conseguirlo es preciso proporcionar a la computadora los valores de ciertos parámetros que se llaman *datos;* a continuación efectuará un cierto número de operaciones sobre estos datos, siguiendo un cierto esquema que le hará funcionar adecuadamente, bien de una sola vez o a petición (mediante un programa). Se obtienen unos *resultados* que permiten explotar y obtener la fiabilidad de la computadora. El esquema podría ser:



En este esquema, el hombre interviene para *alimentar* la máquina (introducir datos) y recoger resultados, así como en la concepción del programa.

Teóricamente una computadora tiene un cierto carácter de universalidad, siendo el programa quien le da su destino propio. También en cada dominio de utilización (científico, gestión comercial y financiera, etc.) la información manipulada es diferente. En general ella se reduce de modo elemental a:

— *caracteres alfanuméricos:* letras, cifras, símbolos...
— *caracteres numéricos:* números según diversos modos de representación, enteros, reales...

Una de las características esenciales de una computadora es que trata información digital, o sea trabaja en lenguaje binario (convencionalmente una serie de 0 y 1). Así, toda la información debe ser digitalizada (traducida a lenguaje binario). El elemento binario (el bit) es la unidad de cuantificación elemental. La medida de la información se hace en número de bits o en número de *octetos* («bytes», en inglés, grupos de 8 bits). Posteriormente se ampliará este concepto.

Para adquirir o distribuir la información, la computadora es capaz de establecer comunicaciones con el mundo exterior. El entorno externo de una computadora está formado por sistemas periféricos o terminales:

— que permiten el diálogo entre el hombre y la máquina (*teclados, impresoras, pantallas de visualización,* etc.);
— capaces de almacenar importantes cantidades de información: memorias secundarias o auxiliares (*discos, disquetes, cintas magnéticas,* etc.).

Algunas veces existen enlaces especializados que permiten a la computadora dialogar con otras fuentes de información: aparatos de medida, dispositivos electrónicos, otras computadoras alejadas (vía redes de comunicaci ón), etc.



Figura 0.2. Computadora personal IBM/PC.

1.1 Clasificación de las computadoras

En primer lugar las computadoras se dividen en analógicas y digitales. En esta obra sólo nos referimos a las segundas, dado que el campo de aplicación de las primeras es reducido y ya prácticamente no se utilizan.*

La clasificación se puede realizar atendiendo a diferentes parámetros: *capacidad de memoria, tamaño físico, precio,* etc. (La parte de una computadora que almacena la información de modo que se pueda utilizar casi inmediatamente se llama *memoria.* La computadora puede almacenar programas o datos en su memoria.)

No obstante, existe una clasificación casi estándar en cuanto a concepto, no en cuanto a límites, que varían de unos autores o fabricantes a otros.

Se pueden dividir en tres grandes grupos: **macrocomputadoras** («mainframe»), **minicomputadoras** y **microcomputadoras.** (En la actualidad comienza a emplearse el término **nanocomputadora** para definir a las microcomputadoras empleadas en el hogar, juegos, etcétera.)

Las **macrocomputadoras** o **supercomputadoras** son muy grandes, pueden necesitar una o varias habitaciones para alojar sus diferentes partes. Permiten procesar gran cantidad de información que pueden almacenar y utilizar para cálculos. Son extremadamente caras. Algunas instalaciones de grandes computadoras pueden costar decenas, centenas o incluso miles de millones de pesetas.

Las **minicomputadoras** o **pequeños sistemas de gestión** no son, como su nombre indica, realmente pequeñas; pero sí más pequeñas que las macro. Pueden hacer cadi todo lo que pueden hacer las grandes macrocomputadoras pero sin su tamaño. No pueden manejar tanta información ni son tan rápidas, por el contrario no son tan caras.

La tercera clase de computadoras son las **microcomputadoras** o simplemente **micros.** Estas máquinas son conocidas popularmente por **computadoras personales.** Son las máquinas más baratas y los precios parece que disminuyen cada día a medida que nacen nuevos modelos. Las microcomputadoras, sin embargo, son ya muy potentes y cada vez pueden hacer mayores aplicaciones. Son más rápidas y tienen memorias más grandes que las micro o macrocomputadoras de hace unos pocos años.

Ahora es posible comprar una computadora por el mismo precio que se compra un *aparato de televisión, un vídeo* o un *coche utilitario.* Los precios son del mismo rango, e incluso menor, lo que le permite entrar en una tienda, incluso en grandes almacenes, y comprar una computadora. Este fenómeno ha hecho posible que muchas personas hayan accedido al uso de la computadora en su propio hogar o lugar de trabajo.

---

* Las computadoras digitales utilizan para su funcionamiento señales digitales 0 y 1 (bajo, alto/0 voltios, 5 voltios).

Los micros son fáciles de utilizar, son pequeños y no son caros. Hoy día numerosas personas (adolescentes, estudiantes, profesionales) los utilizan para jugar, trabajar, etc. Cuando termine este libro usted será capaz no sólo de utilizar las microcomputadoras, sino también de «programarlas».

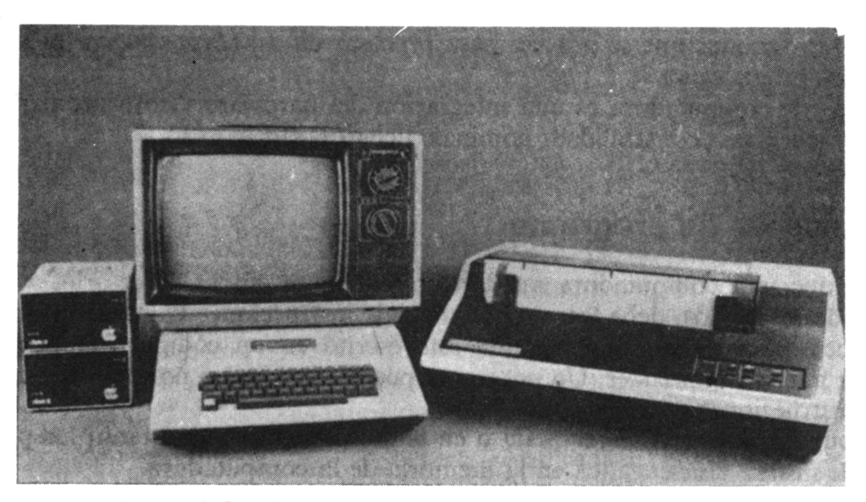


Figura 0.3. Sistema de computadora personal Apple II.

Las microcomputadoras se pueden clasificar en diferentes tipos: **familiares** o **domésticas** (Spectrum, MSX, Amstrad, Commodore 64/128, Dragón 64/200, Enterprise, Einstein, etc.), permiten su uso con un aparato de televisión y pueden realizar juegos, gestión, etc.; **portátiles** (IBM PC, Data General One, Ericcson, Kaypro, etc.), incorporan una pequeña pantalla de vídeo y están dirigidas básicamente a ejecutivos, profesores, viajantes, etc.; las propiamente llamadas **personales** (Apple II, Sinclair, QL, Amstrad CPC6128, Enterprise, etc.), y las **personales profesionales** con representación inequívoca en los diferentes modelos de IBM:PC, XT y AT (PC, siglas de *Personal Computer,* computadora personal) junto a la inmensa saga de compatibles (computadoras con las mismas características hardware y software que permiten ejecutar programas comunes en los diferentes modelos); en el Apéndice V se incluyen características técnicas de diversos equipos de microcomputadoras.



Figura 0.4. Computadoras personales IBM (PC, XT, AT y portátil).

1.2. Hardware y software

El proceso de la información abarca dos partes bien diferenciadas que constituyen toda computadora. En palabras sencillas: lo que se puede tocar y lo que no se puede tocar. Cada parte es muy importante en el trabajo adecuado de la computadora. La parte que se puede tocar se llama **hardware** y la otra **software:**

Hardware: Dispositivos de una computadora o parte física. Circuitería electrónica.
Software: Programas que se utilizan para procesar datos. (*Programador* es la persona que escribe los programas.)

Un sistema de computadora es una integración del hardware y software para realizar aplicaciones específicas: contabilidad, nóminas, ficheros, etc.

1.3. El concepto de programa

Antes de que una computadora pueda leer realmente datos, procesarlos y producir información de resultados, debe leer un conjunto de instrucciones (órdenes a la computadora) llamado **programa,** que indica el proceso requerido. El programa está escrito en un código o lenguaje que la computadora puede comprender. Un programa puede estar hecho por decenas, centenas o millares de instrucciones.

El concepto de *programa almacenado* o en *memoria* significa que tanto los programas como los datos están contenidos en la memoria de la computadora.

Antes de que se pueda escribir un programa se debe indicar la serie de pasos para solucionar el problema. A veces se realizan unos dibujos o gráficos conocidos como *diagramas de flujo* u *ordinagramas.* Los pasos a seguir para diseñar el programa se conocen por *algoritmo.*

2. ARQUITECTURA Y FUNCIONAMIENTO DE UNA COMPUTADORA

Una computadora consiste en una serie de dispositivos que, juntos, funcionan como una unidad integrada o sistema.

Cada computadora consta de los siguientes elementos básicos:

1. Unidades de entrada.
2. Unidad Central de Proceso (UCP).
3. Unidades de Almacenamiento Secundario o Auxiliar (memorias secundarias o auxiliares).
4. Unidad de salida.

Las Unidades de entrada/salida y de almacenamiento secundario o auxiliar se conocen como **periféricos** o **dispositivos.**

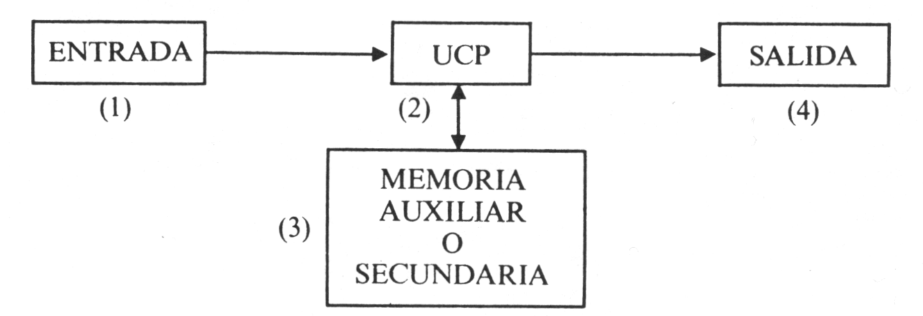


Figura 0.5. Diagrama de bloques de una computadora.

La Figura 0.5 ilustra el diagrama de bloques de una computadora. La Figura 0.6 es una computadora personal típica con algunos de sus elementos periféricos.



Figura 0.6. Computadora HP-150 con unidad central, pantalla y una serie de periféricos conectados a ella.

2.1. La Unidad Central de Proceso

La **Unidad Central de Proceso, UCP,** («Central Processing Unit», CPU en inglés) es el corazón de la computadora, o lo que es igual, el *cerebro* de la máquina. La UCP controla todas las operaciones efectuadas por la computadora y se conecta mediante un cable a cada dispositivo de entrada/salida así como a la memoria auxiliar o externa.

Las operaciones que la UCP realiza son:

1. Lectura de información de un dispositivo de entrada (teclado, unidad de disco, etc.)
2. Procesamiento de datos de entrada que incluye:
   — operaciones aritméticas;
   — comprobaciones y operaciones lógicas.
3. Escritura de información, utilizando un dispositivo de salida (pantalla, impresora, etc.).

UNIDAD CENTRAL DE PROCESO

Los componentes básicos de una UCP se ilustran en la Figura 0.7 y son la Unidad Aritmética y Lógica, la Unidad de Control y la Unidad de Memoria Principal o Central.

1. Unidad Aritmética y Lógica

Controla y ejecuta todas las operaciones aritméticas y lógicas que realiza. Sólo ejecuta cálculos de sumas, restas, multiplicaciones, divisiones y operaciones de relación, como igual (=), mayor que (>), menor que (<), etc.

2. Unidad de Control

Supervisa o comprueba las funciones realizadas por la computadora completa. Un programa de control llamado *supervisor* comprueba todas las operaciones a realizar. El programa propor-

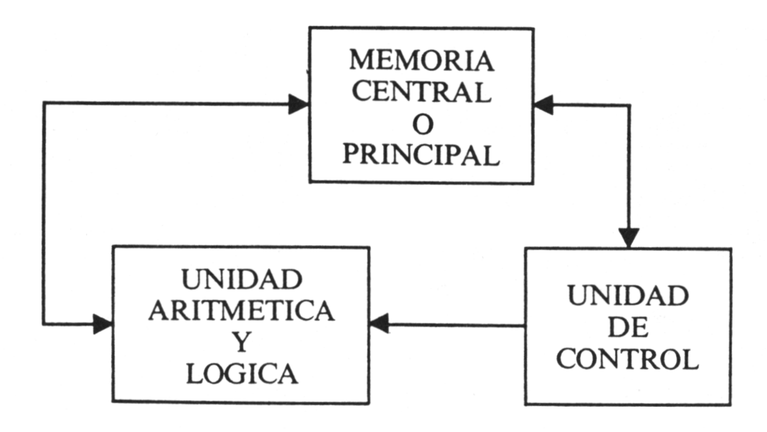


Figura 0.7. Componentes básicos de una UCP.

cionado por el usuario se lee en la memoria central, pero cada instrucción se ejecuta e interpreta por la sección de control de la UCP.

3. Memoria Central o Principal

El almacenamiento principal se suele conocer como memoria principal o central y más correctamente la *memoria* de la computadora. Es la unidad de la UCP donde se sitúa el programa a fin de ejecutarlo o hacerlo funcionar. De modo similar, los datos deben estar en la memoria central para poder ser procesados.

La memoria principal consiste en posiciones de almacenamiento direccionables conocidas como **octetos** («bytes» en inglés). Cada octeto representa un carácter, esto es, una letra o dígito. Un registro o ficha de un fichero que contenga 150 caracteres requiere 150 octetos de memoria cuando se almacene en la computadora.

La capacidad de almacenamiento principal de una computadora se llama **tamaño de la memoria.** El tamaño de la memoria de las microcomputadoras puede ir desde 32.000 a 640.000 octetos, que se representan por 32K o 640K, donde **K** (Kilo-octeto) es aproximadamente igual a 1.000 octetos (1K es igual a 2^{10}=1.024 octetos). Las grandes computadoras tienen capacidades de almacenamiento que sobrepasan el millón de octetos, llamado comúnmente **M, megaocteto** (1M $\cong 2^{10}$=1.048.576 octetos). Las microcomputadoras suelen tener una memoria comprendida entre 64K y 256K, pudiendo llegar con ampliaciones a 512K, 640K o 1M. Sin embargo, las minicomputadoras, y sobre todo las macrocomputadoras, sobrepasan el megaocteto de memoria central. Representantes de las microcomputadoras de pequeña gestión o domésticas, son todos los micros recientemente aparecidos de 128K: Amstrad 6128, Spectrum 128, Commodore 128, etc. En el campo de las computadoras personales profesionales brillan con luz propia: Apple Macintosh de 512K y el recientemente presentado Macintosh Plus de 1M, junto con el ya mítico IBM PC y los dos últimos éxitos del gigante americano: IBM XT e IBM AT con memorias de 512K y 1M, respectivamente, ampliables hasta 8 y 16M. Los nuevos Atari 520ST y 1040ST con memorias centrales de 512K a 1M pueden suponer un nuevo revulsivo en el mercado microinformático por su magnífica relación precios/prestaciones.

2.2. Dispositivos de entrada/salida

Las computadoras como ya se ha dicho leen datos, los procesan y producen información. Hay una gran variedad de dispositivos o *periféricos* que pueden leer datos de entrada y de modo similar presentar los resultados de salida de una computadora.

Los dispositivos más usuales son: *teclados, pantallas, terminales e impresoras, unidades de disco, unidades de casete, «plotters», lápices ópticos, ratones, palancas de mando,* etc.

2.3. Dispositivos de memoria secundaria o auxiliar

Proporcionan a la UCP una capacidad de almacenamiento adicional que le permite leer o escribir informaciones que se pueden conservar durante largo tiempo. Por su carácter universal y limitado es imposible que la memoria principal sirva para almacenar informaciones (programas o datos) durante más tiempo que el necesario para ejecutar un programa. Se necesitan por consiguiente memorias auxiliares que pueden almacenar información de modo permanente o casi permanente.

Los dispositivos de almacenamiento exterior o auxiliar se conocen como **memorias de masa,** por su gran capacidad de almacenamiento en cintas o discos magnéticos, y se denominan *unidades* o *dispositivos* de cinta o disco magnético.

3. ESTRUCTURA FISICA DE LA COMPUTADORA

No existe acuerdo para denominar al componente principal de la computadora, de modo que cuando se emplea este término lo mismo nos podemos estar refiriendo a la Unidad Central de Proceso (o simplemente Unidad Central) o al sistema completo, es decir, la UCP y todos sus periféricos. No obstante, existe cierta tendencia a llamar computadora exclusivamente a la Unidad Central, y así lo interpretaremos aquí.

Cuando se escribe un texto, mediante teclado, aunque físicamente aparece el mismo en la pantalla, se envía a la memoria de la computadora que está en una placa o tarjeta de circuito impreso en la caja o alojamiento de la UCP.

La tarjeta de memoria es una de las varias tarjetas de la computadora, cada una realizada con *chips* o *circuitos integrados* montados en placas o tarjetas de *circuitos impresos.*

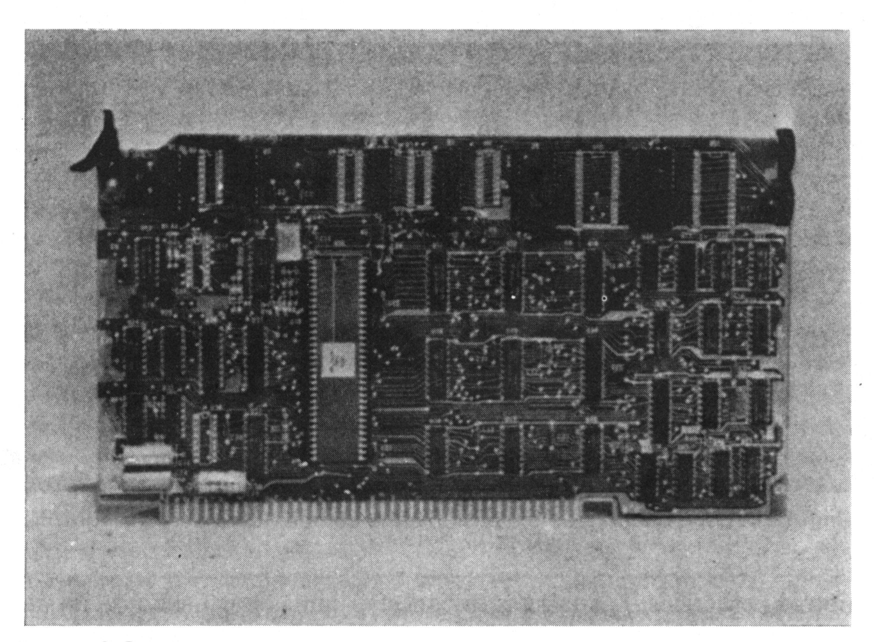


Figura 0.8. Tarjeta de circuito impreso de una microcomputadora IBM/PC.

La memoria de la computadora usualmente se aloja en una tarjeta y a veces en más de una. La Unidad Central de Proceso (UCP) normalmente se aloja en una *tarjeta* llamada *UCP,* que suele contener uno o varios chips que alojan al microprocesador y sus circuitos asociados. Además de estas tarjetas suele haber otras para controlar las unidades de disco, la pantalla o la impresora. Sin embargo, es posible ponerlas todas en una sola placa o tarjeta matriz *(motherboard).*

Algunas computadoras son ampliables, esto es, tienen *ranuras* («slots» en inglés) vacías, donde pueden ponerse nuevas tarjetas, de modo que se puedan conectar ampliaciones de

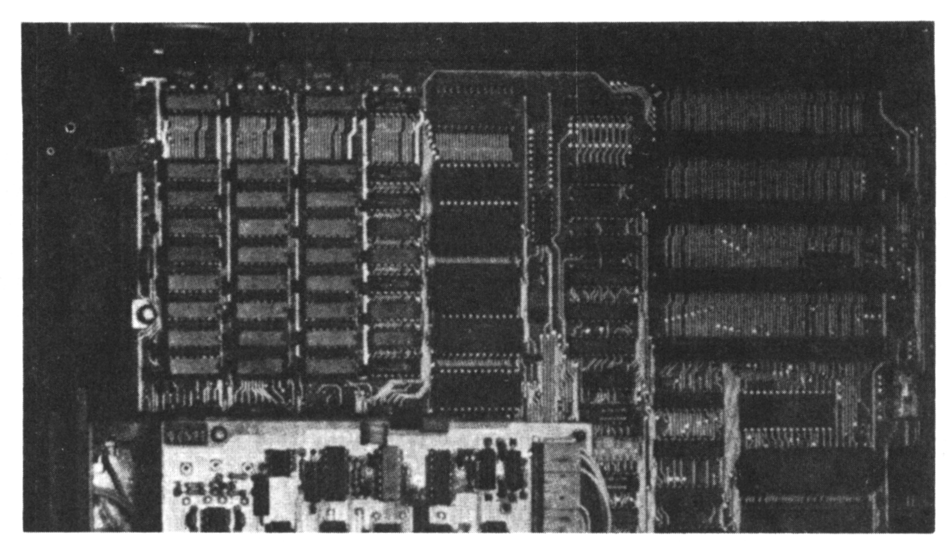


Figura 0.9. La tarjeta del sistema de la computadora IBM/PC con el microprocesador 8088 y la memoria.

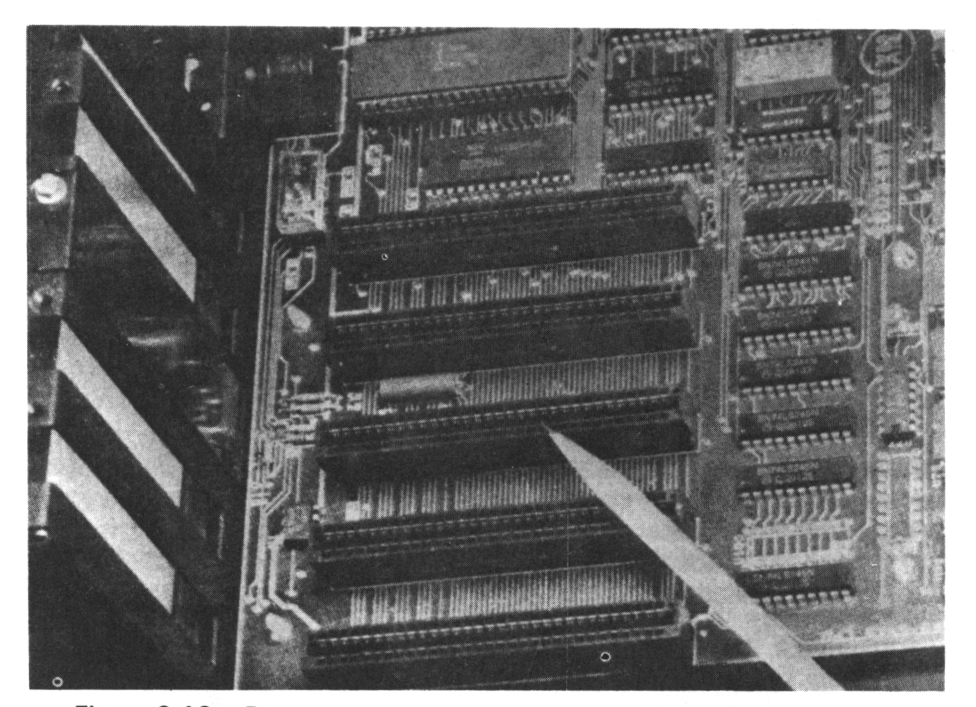


Figura 0.10. Ranuras de expansión en la computadora personal IBM/PC.

memoria u otros periféricos. Las tarjetas se conectan juntas por medio de un *bus* (conjunto de hilos en paralelo que forman un cable de conexión y que permite transmitir información digital por los mismos). Los buses más usuales, S-100, IEEE 488, RS-232C, CENTRONICS, se conocen también como *interfaces* entre la UCP y los periféricos. Las máquinas que tienen el mismo bus pueden utilizar generalmente las mismas tarjetas, lo que significa que las pueden cambiar de una máquina a otra, incluso de fabricantes diferentes.

4. REPRESENTACION DE DATOS EN LA UCP

Todas las computadoras utilizan combinaciones de números en el sistema de numeración binaria para representar cada **carácter**, donde un carácter se define como una letra, dígito,

símbolo de puntuación u otro símbolo. Los caracteres en blanco son caracteres como lo son los símbolos que le indican a la computadora que pase a la siguiente línea o página.

En el **sistema de numeración binario** hay sólo dos dígitos posibles: 0 y 1. Este sistema es ideal para el proceso de la computadora debido a que el «1» se utiliza para representar la presencia de un impulso o señal eléctrica en la circuitería de la computadora y un «0» se utiliza para representar la ausencia de tal señal.

La Figura 0.11 ilustra cómo utiliza la computadora el sistema binario.

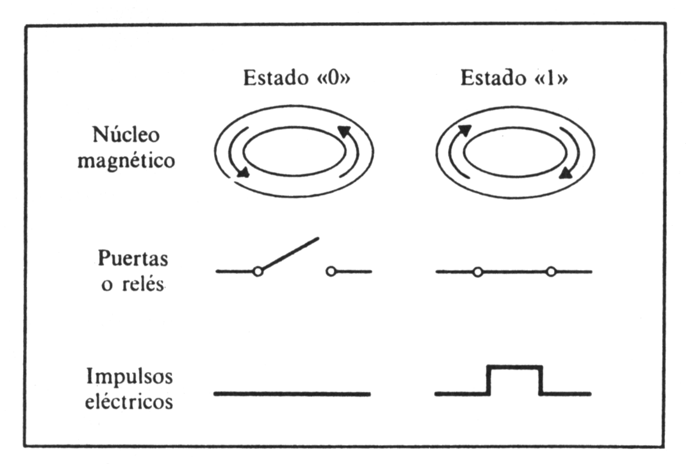


Figura 0.11. Uso del sistema binario por la computadora.

Cada dígito decimal puede ser representado utilizando números binarios:

Decimal	*Binario*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
.	.
.	.

Cada dígito binario o **bit** se representa por uno de los 2 posibles estados: 0 o 1. Su valor decimal equivalente (peso) depende de la posición que ocupe.

... b_3	b_2	b_1	b	Dígitos binarios de un número
... 2^3	2^2	2^1	2^0	Peso dígitos binarios
... 8	4	2	1	Valor decimal

11 en binario, por ejemplo, es equivalente al número decimal 3 // $11=1\times2^1+1\times2^0=2+1=3$.

Valor exponencial	2^1	2^0
Valor decimal	2	1
Valor binario	1	1

11 en binario indica la presencia del bit b_1 (2^1) y el bit b_0 ($2^0=1$); por consiguiente es igual al número decimal 3 ($2^1+2^0=2+1=3$).

Además del sistema binario, las computadoras pueden trabajar con los sistemas de numeración octal y hexadecimal. En el Apéndice III se incluye una breve explicación de los tres sistemas así como tablas de conversión numérica para pasar de un sistema a otro.

Los caracteres se representan por *octetos* («bytes» en inglés) que son números binarios de 8 bits. Un octeto se utiliza para almacenar un único carácter. Una computadora no puede comprender los caracteres escritos por teclado y es necesario convertirlos a números binarios, utilizando normalmente el código ASCII (siglas de American Standar Code for Information Interchange, «Código normalizado americano para intercambio de información»). *Véase* el Apéndice I para ampliación del tema. Existen 128 o 256 (2^7 o 2^8) caracteres diferentes, según se considere el código ASCII básico o el ampliado.

Las unidades de medida del tamaño o capacidad de la memoria son 1K y 1M (1K, abreviatura de 1Kocteto o 1Kilocteto, representa 1.024 octetos, aproximadamente un millar, —2^{10}=1.024— y 1M representa sobre un millón de octetos, 1.048.576 —$2^{10}\times2^{10}$— octetos).

La memoria central de una microcomputadora se suele expresar en K y suele ser del orden de 64K a 256K.

Otro concepto importante a considerar es la **palabra,** conjunto fijo de bits, aunque normalmente suelen ser 8, 16, 32 o 64 bits. Del tamaño de la palabra resulta una clasificación de las computadoras por su longitud, esto es, de 8, 16, 32 o 64 bits. Las microcomputadoras domésticas más usuales tienen 8 bits (Spectrum, MSX, Commodore 64, Amstrad, AppleII, etc.) y las computadoras personales o profesionales tienen 16 bits, tales como IBM PC, XT, AT, Apple Macintosh, Ericcson PC, Sperry PC, etc., e incluso algunas de 32 bits.

5. MEMORIA CENTRAL O PRINCIPAL

La memoria central se puede clasificar de dos formas distintas según:

1. EL MATERIAL DE QUE ESTAN CONSTITUIDAS

Memoria del núcleo magnético

Es una tecnología basada en la imantación de núcleos o anillos magnéticos; según se magnetice en una dirección o en otra, almacenará un «1» o un «0». Fueron las memorias utilizadas antes de la aparición de las memorias de semiconductores o circuitos integrados. Hoy ya no se fabrican, aunque todavía quedan computadoras funcionando con esa técnica.

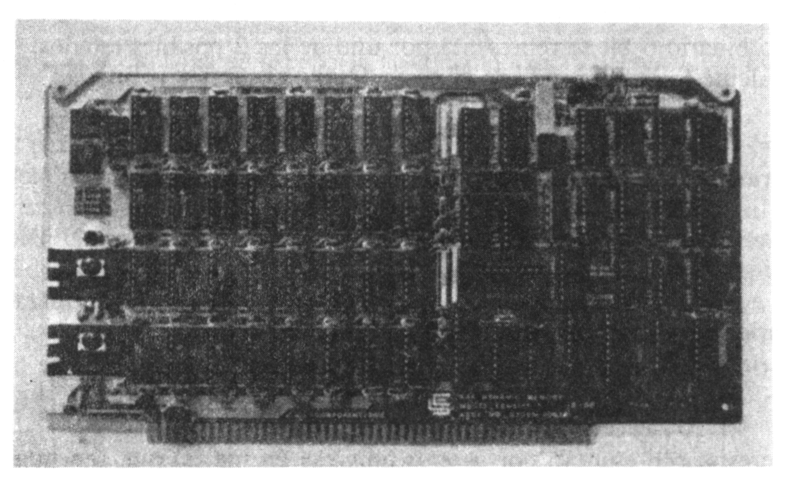


Figura 0.12. Tarjeta de circuito impreso con diversos circuitos integrados.

Circuitos integrados

Las computadoras más modernas utilizan memorias consistentes en circuitos integrados (centenares e incluso millares de componentes electrónicos alojados en una delgada oblea de silicio). Existen circuitos integrados que pueden almacenar hasta 64K bits o más en un solo «chip» o «cápsula».

Memorias de burbuja magnética

Existe un nuevo desarrollo de memorias que ya comienza a utilizarse y que puede reemplazar en unos años a los circuitos integrados, son las denominadas **memorias de burbuja magnética**, ofrecen la ventaja sobre otros tipos de memoria de que los datos pueden permanecer en la memoria aunque se corte la alimentación y además contiene una gran capacidad de almacenamiento. Tiene el inconveniente de ser aún muy cara, pero cuando se consiga reducir el precio será un excelente producto.

2. LA FUNCION QUE REALIZAN

La memoria central se divide generalmente en dos clases diferentes: RAM y ROM.

RAM. Memoria de Acceso Aleatorio o Directo

RAM, siglas de Random-Acces Memory (Memoria de Acceso Aleatorio o Directo), es la memoria que se utiliza para almacenar programas y datos. Se puede acceder a ella y modificar su contenido tantas veces sea necesario. La información que se pone en ella se pierde cuando la computadora se apaga (a menos que conserve la información en una memoria auxiliar).

Es una memoria *volátil* ya que su contenido se destruye cuando la alimentación eléctrica se interrumpe.

En la memoria RAM se puede *leer* y *escribir* información mediante los oportunos programas y por ello se llama Memoria de lectura/escritura.

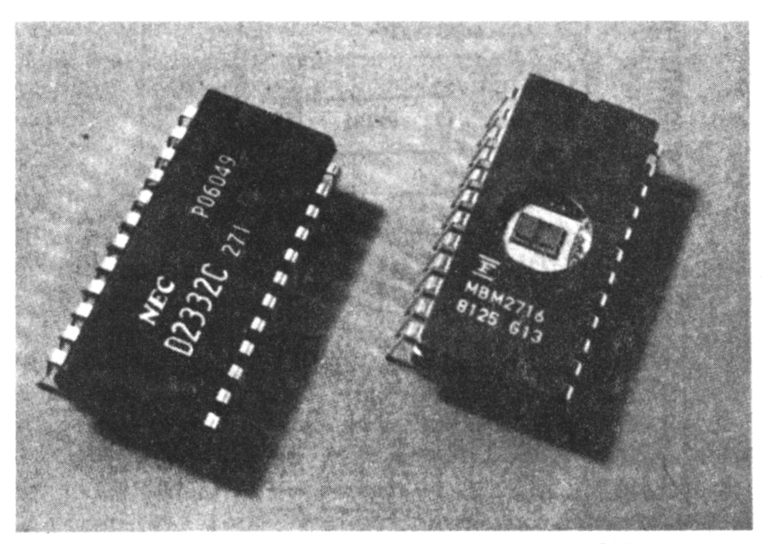


Figura 0.13. Chips de memoria RAM y ROM.

ROM. Memoria de sólo lectura

ROM, siglas de Read-Only Memory (Memoria de sólo lectura), es la parte de memoria principal que contiene los procedimientos y funciones de control que no pueden ser modificados por el programador. Las funciones en la memoria ROM están almacenadas permanentemente y no pueden modificarse por programa. Sólo se puede acceder a ella para *lectura* de

información y nunca para escribir. Es una memoria muerta de sólo lectura y *no volátil;* su contenido no se modifica ni destruye en caso de no alimentación eléctrica.

PROM. Memoria programable

PROM, siglas de Programmable ROM (ROM programable), es una memoria ROM que puede ser programada por el usuario o el fabricante, pero una vez programada ya no se puede modificar su contenido.

EPROM. Memoria programable y borrable

Siglas de Erasable Programmable ROM (ROM borrable y programable). Es una memoria ROM que se puede borrar (eliminar la información que tenga almacenada) y volver a programar de nuevo con información diferente. Actúa como una ROM cuando está programada.

La memoria principal de una computadora consta de memorias RAM y memorias ROM en cantidad y proporción variable, según la aplicación.

La memoria RAM contiene datos y programas introducidos en la microcomputadora durante su funcionamiento, a partir de los periféricos por medio de las unidades de intercambio de comunicaciones. Son los programas de los usuarios que responden a diversas aplicaciones. Es la memoria central o interna de la computadora y es su característica más sobresaliente. La memoria ROM contiene programas permanentes o inalterables que no son susceptibles de modificar y son los programas bases de la computadora. Suelen contener los lenguajes intérpretes o compiladores, así como el sistema monitor y operativo de la máquina.

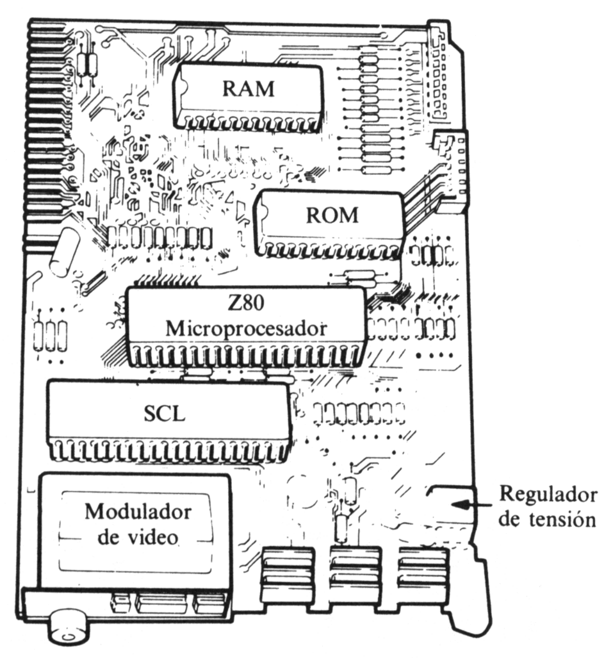


Figura 0.14. Tarjeta de la histórica microcomputadora ZX-81 de Sinclair.

6. ENTRADA Y SALIDA DE DATOS

¿Cómo «hablar» con la computadora? ¿Cómo introducir información a la máquina? *Entrada* es la operación de introducir información o alimentar la computadora con datos.

¿Cómo «extraer» los datos de la computadora una vez procesados? La información procedente de la computadora se denomina *salida.*

Los dispositivos de hardware que se utilizan para comunicarse con la computadora se llaman de entrada/salida o E/S. Estos dispositivos se conocen también por *periféricos* de E/S y se conectan a la Unidad Central mediante los cables o conexiones adecuadas.

Dispositivos de entrada	Teclado Unidad de casete Unidad de disco Lápiz óptico, palanca de mando («joystick»), ratón Digitalizador
Dispositivos de salida	Pantalla Impresora «Plotter» (registrador o trazador gráfico) Unidad de disco o cinta



Figura 0.15. Computadora AMIGA de Commodore, con teclado, pantalla, unidad de disco y ratón.

6.1. Teclado

El teclado es el dispositivo más utilizado en computadoras. Es similar al teclado de las máquinas de escribir eléctricas o de los teletipos añadiendo algunas teclas adicionales.

El teclado comprende un conjunto de teclas, de modo que la pulsación de una de ellas transmite a la unidad central una información codificada que representa al carácter en un código determinado, normalmente ASCII.

Existen diversos tipos de tecnología para las teclas del teclado: *mecánico, sensibles al tacto,* etc. Los teclados de teclas mecánicas son los más económicos, estando garantizados contra millones de pulsaciones. Una característica muy notable en la actualidad es la *repetitividad* de las teclas, o lo que es igual aquellas que poseen esta propiedad, mientras estén pulsadas, el carácter correspondiente se repite a una cierta cadencia hasta que se suelte dicha tecla. El ejemplo típico de teclas repetitivas son la barra espaciadora y las teclas de movimiento del cursor. Esta característica puede estar disponible para una, varias o todas las teclas del teclado.

Normalmente los teclados constan a su vez de diferentes teclados parciales:

— *Teclado principal:* Agrupa las teclas típicas de una máquina de escribir, letras, números, signos de puntuación, etc.
— *Teclado numérico:* Contiene los diez dígitos (0 a 9) y algunas teclas de funciones, como +, −, *, /, etc.
— *Teclas de funciones programables:* Contiene diversas teclas que pueden ser programadas por el usario, se representan normalmente por F_0, F_1, F_2, ... o bien K_0, K_1, K_2, etcétera.

Existen dos grandes tipos de teclados principales según la organización de los caracteres alfanuméricos sobre las teclas:

— El teclado QWERTY es el más universal y extendido. Se utiliza en América y Europa, incluyendo España.
— El teclado AZERTY es el utilizado en los países de lengua francesa.
— El teclado DVORAK, que comienza a implantarse en USA, y tal vez pronto en el resto del mundo.

Cuando elija un teclado observe si dispone de letras mayúsculas, minúsculas, letras especiales, por ejemplo la ñ, etc. Actualmente, por suerte para el usuario informático español, el Ministerio de Industria exigirá a los fabricantes de computadoras la normalización de teclados al alfabeto castellano, e incluso nos figuramos que a otros alfabetos, como en el caso del catalán.

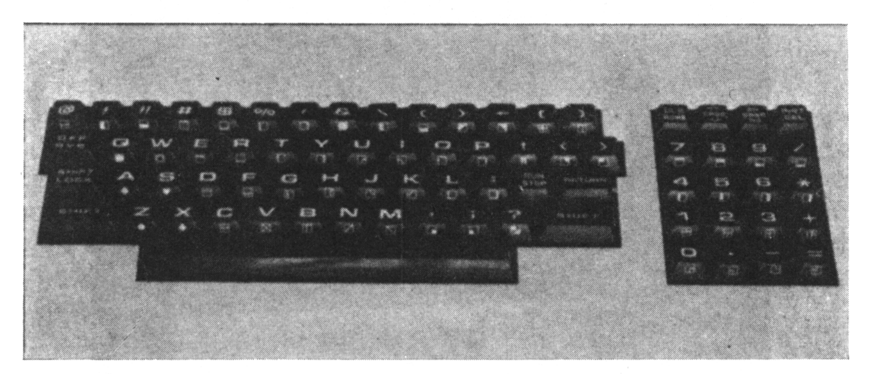


Figura 0.16. Teclado con teclas numéricas independientes.



Figura 0.17. Teclado sin teclas numéricas independientes (Apple II).



Figura 0.18. Teclado estándar IBM/PC y compatibles.

6.2. La pantalla de visualización

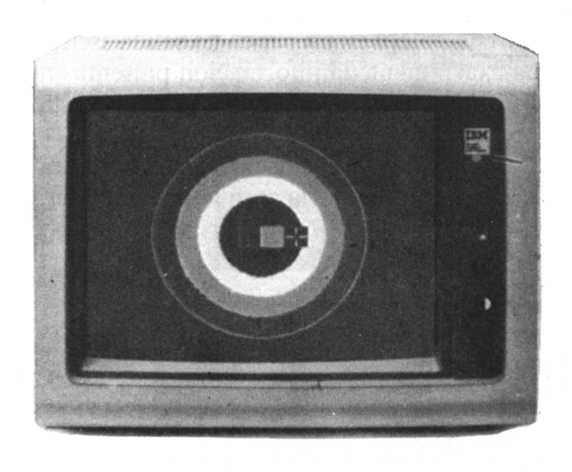
La pantalla de visualización es el dispositivo de salida más utilizado en informática. Esta unidad puede ser un simple *aparato de televisión,* blanco/negro o color, o bien un *monitor de vídeo,* que es un tubo de rayos catódicos (TRC) con una presentación similar a una pantalla de televisión. La salida se visualiza instantáneamente en la pantalla.

La salida de presentación visual puede estar en formato escrito (modo texto) o en formato gráfico (modo gráfico).

El tamaño de la pantalla está definido por su diagonal y expresado en pulgadas, usualmente 12 y 15 (o sea, 30 y 38 cm). Recientemente han aparecido las micropantallas de 5 y 9 pulgadas incorporadas a las computadoras portátiles como IBM PC o KAYPRO.

Las pantallas pueden ser *monocromáticas* y de *colores,* siendo las más frecuentes las primeras, en donde el color del carácter visualizado depende únicamente del tipo de fósforo depositado en la cara interna de la pantalla. Los colores más frecuentes son blanco sobre fondo negro, verde sobre fondo negro y ámbar. Algunos sistemas permiten la inversión vídeo del carácter visualizado (similar al negativo de la fotografía), con lo que los colores se convierten en negro sobre fondo blanco o negro sobre fondo verde.

Las pantallas más modernas permiten la visualización de caracteres alfanuméricos y gráficos. La representación de dichos caracteres se realiza mediante matrices de pequeños puntos llamados *pixels* y de dimensiones 7 × 9, 10 × 12 puntos, etc.



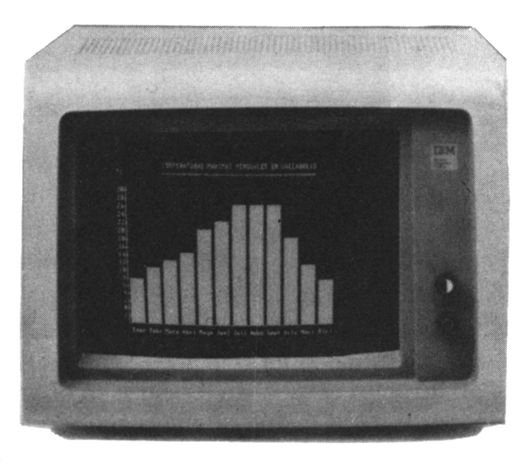


Figura 0.19. Monitores monocromo y color de la computadora personal IBM/PC.

6.3. Terminal

Se denomina *terminal* al dispositivo que integra en una sola unidad el teclado y la pantalla de visualización, y que suele estar conectado a una computadora central. Los terminales pueden ser «inteligentes» o «tontos» según tengan, o no, unidad central de proceso propia para realizar tareas independientes.

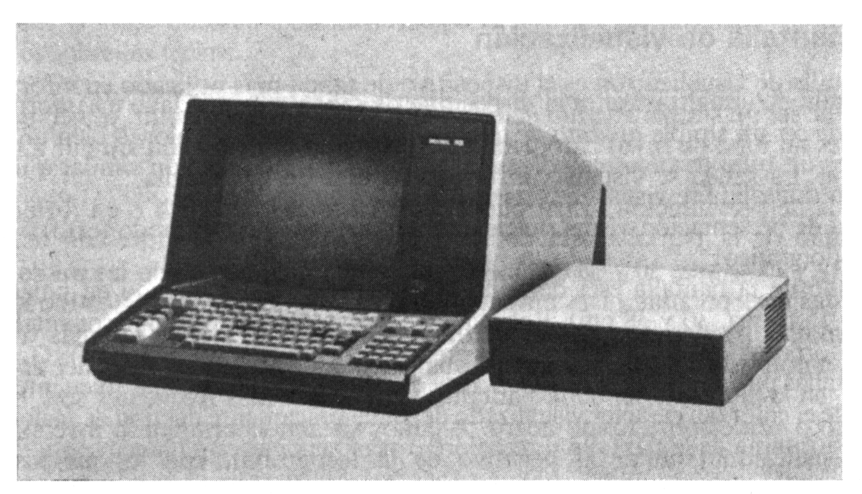


Figura 0.20. Terminal de computadora.

6.4. Impresoras

La unidad o dispositivo de impresión se denomina **impresora** y permite proporcionar al usuario de una computadora una salida de información no volátil en un papel. La salida escrita se conoce como *hard copy* (copia dura), si representa exactamente la información presente en la pantalla; en muchas computadoras existe una tecla HARD COPY, o COPY que realiza esta función nada más pulsarla.

La gama de impresoras en el mercado es muy grande y van desde la pequeña impresora que copia 60 a 90 caracteres (letras, números, etc.) por segundo *(cps)* a la gran impresora láser, que copia 10.000 líneas por minuto *(lpm)*.

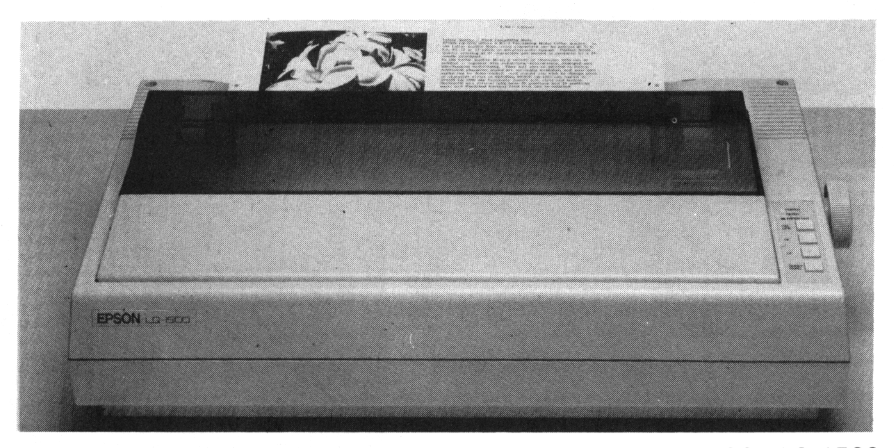


Figura 0.21. Impresora profesional de alta calidad de letra, EPSON LQ-1500.

Las técnicas de impresión más usuales son *Impacto, Sin impacto* y estas últimas a su vez *electrostáticas, chorro de tinta, electrolíticas, xerográficas* o a *láser*. En general, las impresoras sin impacto son más fiables y de mejores prestaciones.

El modo de impresión de los caracteres define las tres familias de impresoras:

— *De caracteres:* Se imprimen unas letras detrás de otras después del desplazamiento y posicionamiento de la cabeza de impresión. Velocidad de 10 a 1.000 *cps*. Se conocen también como impresoras serie.

— *De líneas:* Todos los caracteres de una línea se imprimen simultáneamente. Su velocidad puede ir de 300 a 10.000 *lpm*. Se conocen también como impresoras paralelo.

— *De páginas:* Son impresoras de muy altas prestaciones y que se utilizan en equipos grandes que imprimen páginas completas, por técnica xerográfica y láser. La velocidad de impresión puede ir de 100 a 1.500 páginas por minuto (1 página = 50 o 60 líneas).

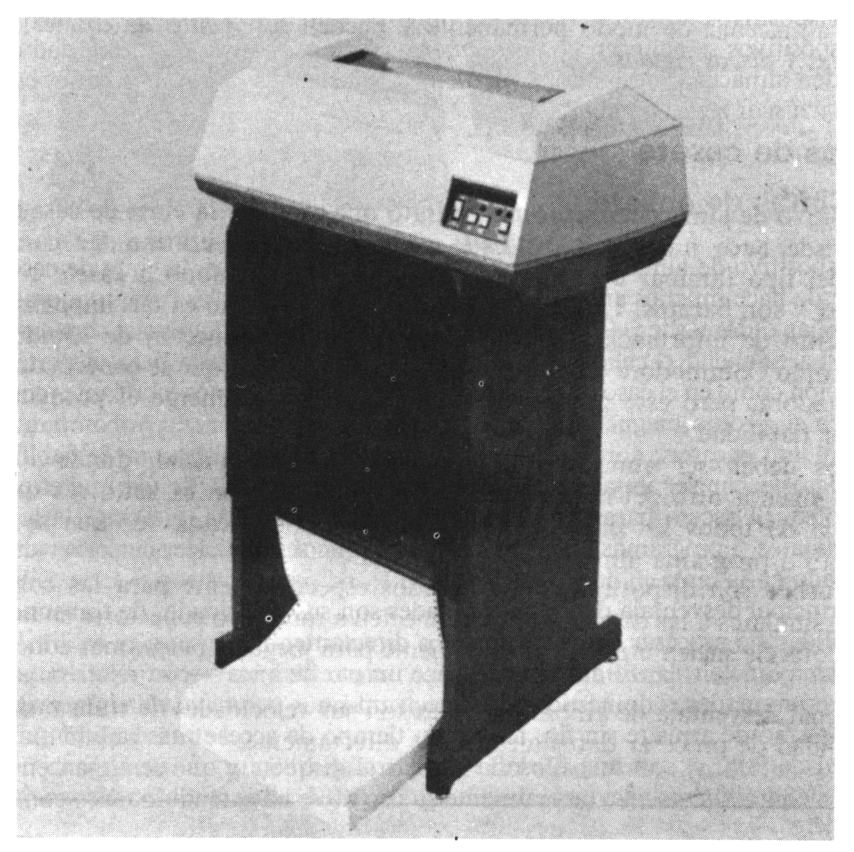


Figura 0.22. Impresora de líneas.

De acuerdo con la velocidad de impresión se clasifican en:

• *muy baja velocidad*	10 a 30 cps
• *baja velocidad*	30 cps a 100 cps
• *media velocidad*	100 lpm a 600 lpm
• *alta velocidad*	600 lpm a 2.500 lpm
• *muy alta velocidad*	más de 2.500 lpm

El formato del papel que pueden utilizar suelen ser estándar y contener 80 o 132 columnas en una línea.

La alimentación y arrastre del papel puede permitir *papel continuo* tipo acordeón o pijama, *página a página*, con carga (introducción) manual o automática. Las posibilidades técnicas de alimentación del papel están relacionadas con su modo de arrastre: *fricción* (la rotación de un cilindro permite el avance por fricción del papel y por consiguiente hoja a hoja), *tractor* (la máquina lleva dos ruedas dentadas que permite arrastrar papel continuo).

Los criterios que se deben seguir para seleccionar una impresora son:

1. Calidad requerida para las letras.
2. Velocidad.
3. Tamaño y tipo del papel requerido.
4. Caracteres de impresión disponibles con la impresora (esto es, si necesita símbolos, gráficos, alfabeto especial, etc.).
5. Posibilidad de gráficos si éstos son necesarios.
6. Fiabilidad.

7. DISPOSITIVOS DE ALMACENAMIENTO EXTERNO (MEMORIAS DE MASA)

Los dispositivos de almacenamiento ponen y toman información de medios magnéticos. Estos dispositivos se conocen como *memorias de masa* por la gran cantidad de información que pueden almacenar de modo permanente y pueden ser *cintas de casetes, cartuchos, discos flexibles y discos rígidos.*

7.1. Cintas de casete

El dispositivo de almacenamiento más barato que existe es la cinta de casete utilizada en audio desde hace muchos años. Este medio es el que utilizan las computadoras personales del tipo familiar o doméstico. Cualquier magnetófono o casete comercial se puede utilizar y son baratos. La fidelidad o calidad de audio no es tan importante para el almacenamiento de información como en el caso de reproducción de sonido. Algunos fabricantes como Commodore ofrecen su propio magnetófono que se conecta directamente a la computadora, pero este procedimiento encarece normalmente el producto aunque ofrece mayor fiabilidad.

Los datos deben ser almacenados en un formato secuencial, donde 1 octeto de información sigue a otro. El acceso para obtener información es lento, ya que exige la lectura previa de todos los datos o programas anteriores cada vez que se desea leer cualquier dato o programa almacenado en la cinta.

La principal desventaja de ambas unidades son sus velocidades de transmisión lentas y su imposibilidad de procesar datos aleatorios o directamente.

La firma Sinclair, lanzó al mercado hace un par de años —con relativo éxito temporal— para sus computadoras domésticas ZX Spectrum unos *cartuchos* de cinta magnética que con un mecanismo de arrastre sin fin, tenían un tiempo de acceso más rápido que la cinta y una capacidad de 90K y, con una filosofía similar al disquete y que se alojan en unas unidades conocidas como *microdrives* (actualmente su uso no se ha extendido tanto como sus creadores esperaban).

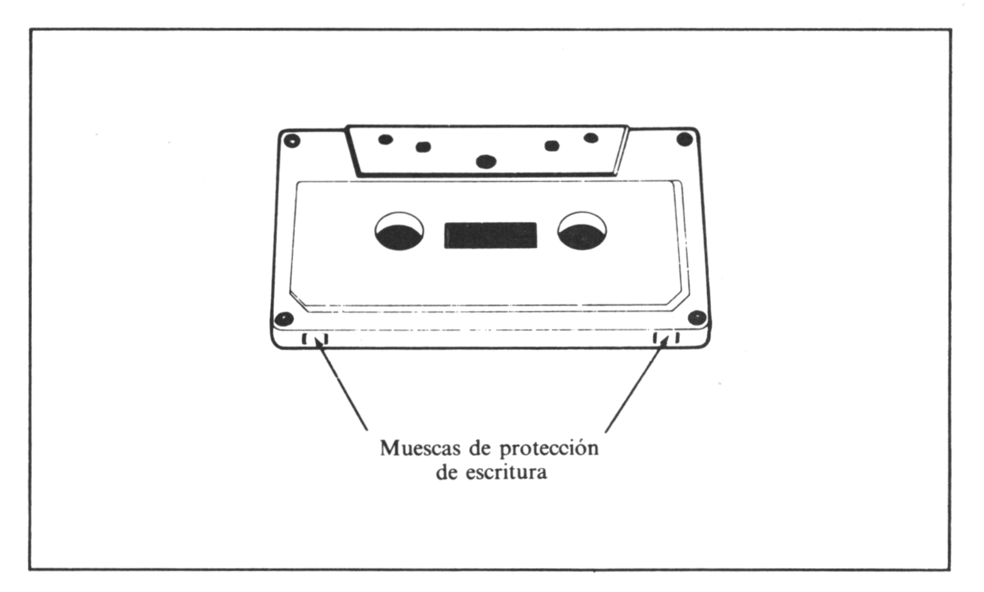


Figura 0.23. Cinta de casete.

7.2. Discos flexibles

El disco flexible o disquete (*«floppy»* en inglés) es el medio de memoria auxiliar más frecuente en las computadoras personales. Son discos fabricados en material plástico recubierto de una capa de óxido magnético de espesor muy delgado. El disco está encerrado en una funda protectora para impedir doblamiento, rayado y contaminación durante su manejo.

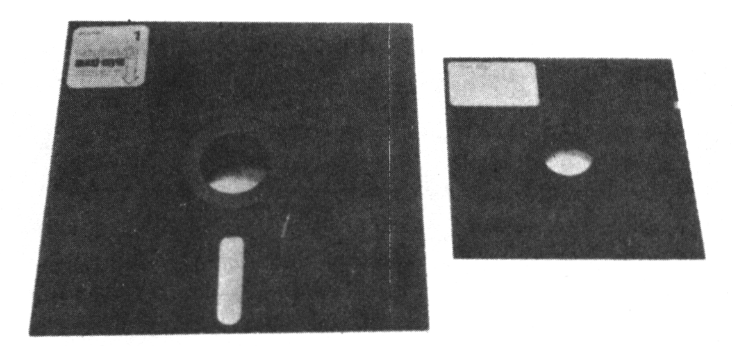


Figura 0.24. Discos flexibles de 8 y $5^1/_4$ pulgadas.

Permite almacenamiento de datos, tanto aleatorio o directo como secuencial, por lo que el tiempo de acceso a una unidad de disco flexible es muchísimo más rápido que a una casete o cartucho.

Los discos flexibles se presentan en tres versiones según el diámetro del disco: *Disquetes* de 8 pulgadas (20 cm), *minidisquetes* de 5 1/4" (13 cm) y *microdisquetes* de 3 1/2" (8 cm) o de 3" (7 cm) que han aparecido recientemente y que ya han adoptado computadoras como Apple Macintosh, HP-150 (3 1/2 "), Amstrad (3 "). Sinclair lanzó hace un par de años al mercado para sus populares computadoras ZX Spectrum y QL unos *cartuchos magnéticos* similares a los disquetes y que se cargan en *Microdrives* (microunidades de disco), pero con relativa poca suerte.

Los discos flexibles se pueden grabar en una sola cara o en las dos caras, llamándose en consecuencia de una **sola cara** o de **doble cara.** Asimismo la cantidad física de grabación de datos puede ser con densidad normal o doble, conociéndose entonces por discos flexibles de **simple densidad** o **doble densidad.**

Según el tipo de disquete se puede llegar a almacenar desde 140K (como es el caso del Apple II), 360K en las computadoras IBM PC y compatibles y hasta 1.2M en las nuevas IBM AT.

Sin embargo, muchos usuarios encuentran que los discos flexibles tienen una capacidad de almacenamiento que es inadecuada. Incluso discos de doble cara, doble densidad que pueden almacenar hasta un millón de caracteres a veces no basta. Ya que una página de texto escrito contiene aproximadamente 4.000 caracteres, un disco flexible de 1Mega podría almacenar como mucho 250 páginas de datos, que puede no ser suficiente para algunas aplicaciones, particularmente donde se requieran grandes ficheros.

Como resultado de ello han aparecido hace unos años *los discos rígidos,* (tecnología Winchester) disponibles para minicomputadoras y microcomputadoras, cuya capacidad de almacenamiento puede ir desde los 10M de la computadora IBM XT a los 20 M del AT pasando por los 30M del nuevo modelo de AT y los 44M del Sperry XP.

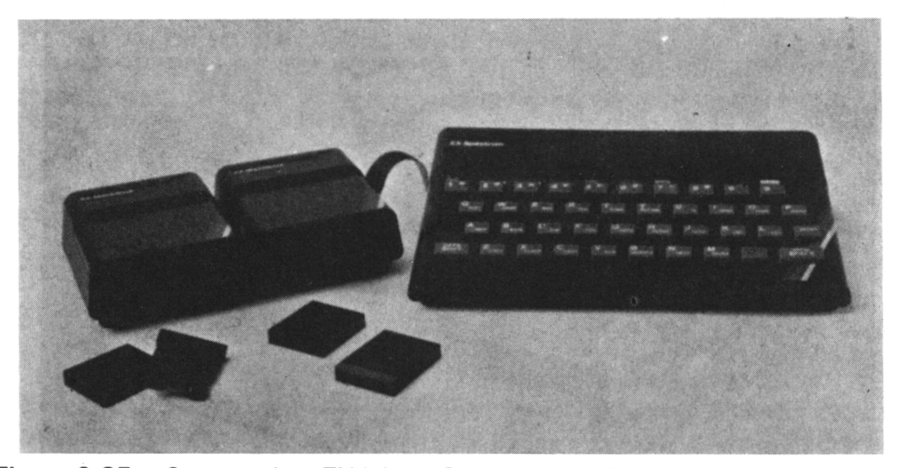


Figura 0.25. Computadora ZX básica Spectrum con 2 Microdrives, 2 cartuchos de memoria externa y sus fundas.

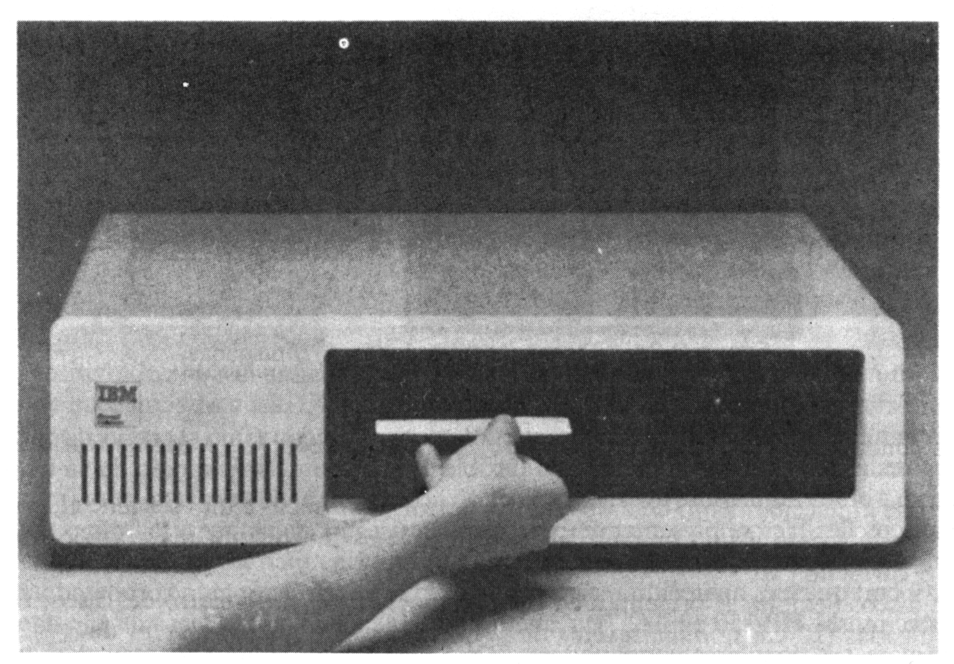
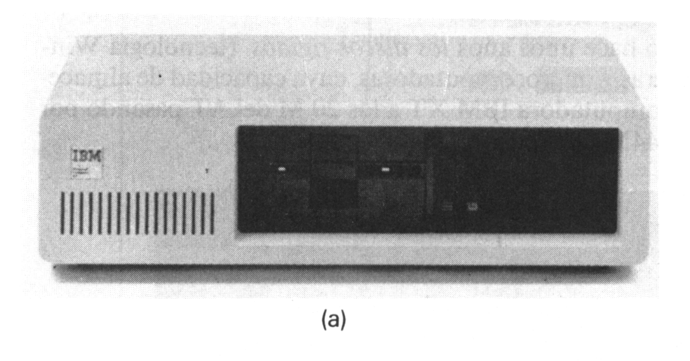


Figura 0.26. Inserción de un disquete flexible en una unidad doble de disco.

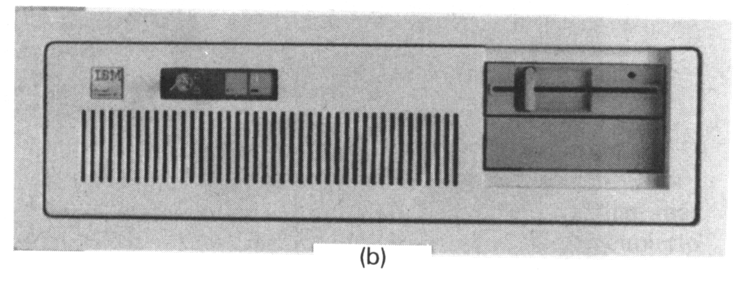
7.3. Discos rígidos

Históricamente, el disco **rígido** o **duro**, en cobre o en aluminio, apareció en el mercado informático antes que el disco flexible de plástico (1956 frente a 1970). En efecto, este dispositivo de memoria secundaria es utilizado tradicionalmente en informática con las computadoras clásicas («macros» y «minis»).

Los discos que se pueden fabricar con metales rígidos, al contrario de los discos flexibles, tienen un soporte que es un material *duro* o *rígido,* de ahí su nombre.



(a)



(b)

Figura 0.27. Unidades de disco flexible y disco rígido fijo: a) IBM XT; b) AT.

Se caracterizan por:

— Diámetro de 14 pulgadas (unos 36 cm).
— Gran capacidad de almacenamiento: de cinco millones a varias centenas de millón de octetos.
— Tiempo medio de acceso muy rápido, 10 a 50 m.
— Velocidad de transferencia de información de uno a cinco millones de octetos/segundo.

Como inconveniente, este disco es muy costoso. Existen dos tipos: *fijos* y *removibles,* según que se puedan intercambiar unos discos por otros o no.

Hacia 1979-1980 han aparecido los discos rígidos de pequeñas dimensiones: 8 y 5 1/4 pulgadas. Esta innovación debida a IBM se conoce como **tecnología Winchester** y permite almacenar una media entre 5 y 70 millones de caracteres, pero existen posibilidades para mayores capacidades. Las unidades de disco Winchester proporcionan gran capacidad de almacenamiento y mayor fiabilidad que otros tipos de unidades de disco.

Los discos Winchester se caracterizan por:

— El disco es fijo (inamovible).
— Las cabezas de lectura/escritura, el carro que los soporta y los platos magnéticos están encerrados en un recinto hermético ausente de contaminación.
— La densidad de información ha aumentado considerablemente.
— La fiabilidad ha aumentado notablemente y el coste de almacenamiento de octetos se ha reducido.

El mayor inconveniente es ser disco fijo e inamovible, lo que lleva consigo la necesidad de realizar copias de seguridad del contenido del disco en otro medio. La mejor solución a este problema consiste en utilizar en unión con el disco fijo, un disco flexible y en el caso ideal de una unidad de cinta para realizar **copias de seguridad** *(backup)* de los datos y programas almacenados en el disco fijo. Estos soportes suplementarios no tienen más función que la conservación o archivo de información. Las microcomputadoras IBM PC, XT, AT y compatibles soportan discos fijos de capacidades de 10M, 20M y 30M, llegando en algunos modelos a 40M, e incluso superiores.

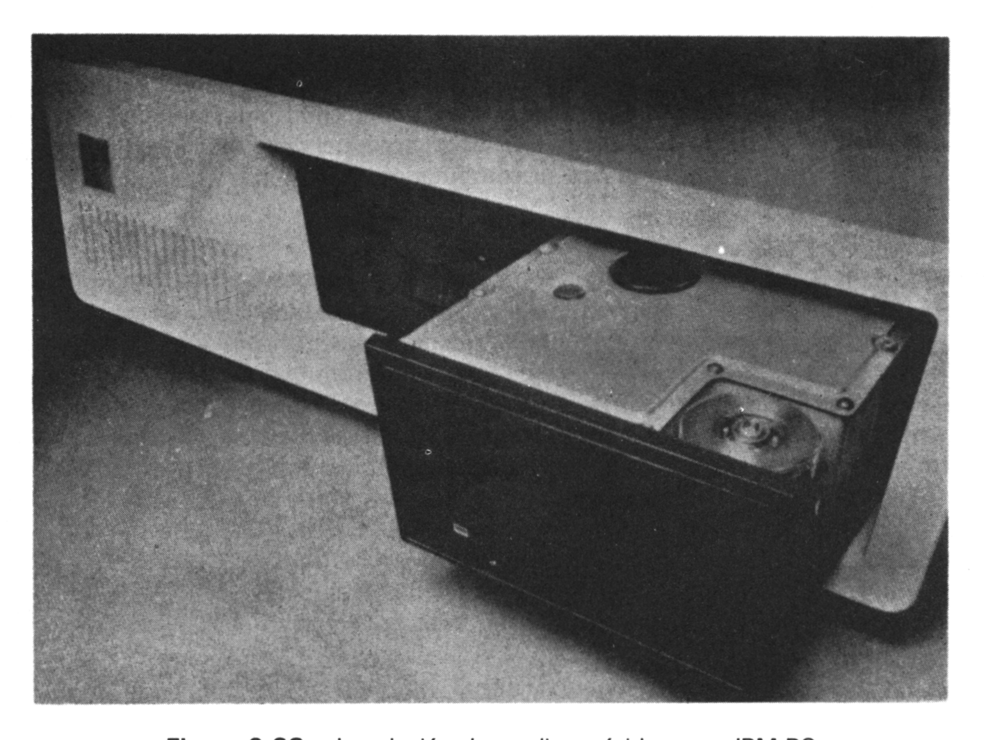


Figura 0.28. Instalación de un disco rígido en un IBM PC.

8. OTROS PERIFERICOS

A la unidad central se le pueden conectar numerosos periféricos entre los que cabe destacar **lápiz óptico, palancas de control** o **mando** *(joysticks),* **digitalizadores, trazadores gráficos** *(plotters),* **sintetizadores de voz, ratones,** etc.

EL LAPIZ OPTICO

Es un periférico de entrada/salida en unión con una pantalla. Es un medio de comunicación potente y cómodo que permite detectar un punto determinado en la pantalla mediante la luz que ella misma emite. Es particularmente útil para la elección de un *menú* (lista de opciones posibles), ya que basta apuntar el lápiz al punto deseado. Otra aplicación usual es en el tratamiento de gráficos. Cada día es mayor su utilización en las computadoras personales.

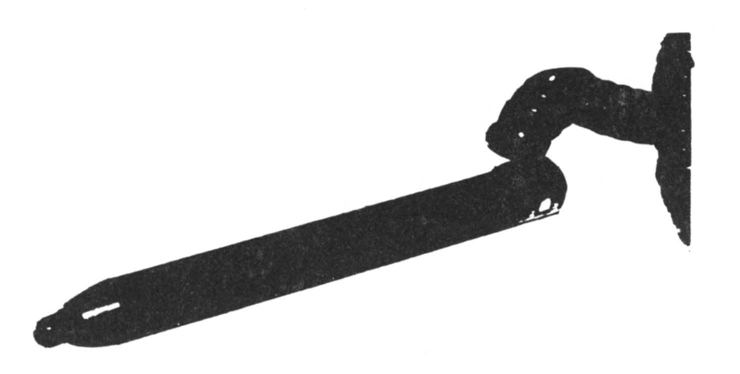


Figura 0.29. Lápiz óptico para la computadora IBM/PC.

PALANCAS DE MANDO (JOYSTICKS) / RATON

Las *palancas de mando* utilizadas para controles de juegos permiten movimientos en todas las direcciones y disponen de un pulsador que se suele utilizar para la acción de *fuego* en los juegos de vídeo. El *ratón* es un dispositivo que se mueve sobre una superficie lisa (mesa) y arrastra un puntero o cursor en la pantalla con su movimiento, diseñado originalmente por XEROX, Apple lo ha popularizado con su computadora Macintosh.

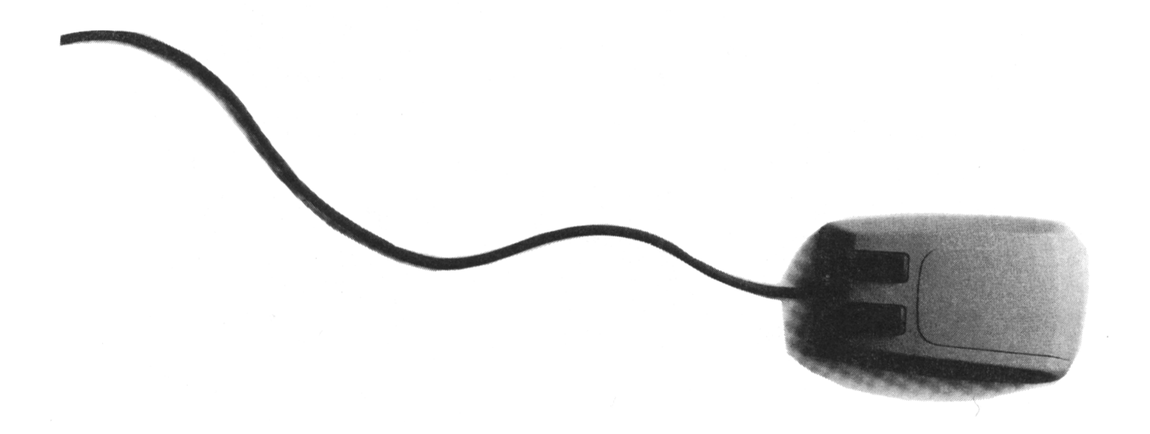


Figura 0.30. Ratón de Microsoft para el IBM PC.

DIGITALIZADORES / TRAZADORES GRAFICOS

Los *digitalizadores* son dispositivos que permiten la digitalización de planos o mapas, introduciendo sus datos en la memoria de la computadora, a medida que una *lupa* o *puntero* se desplaza por las diferentes curvas de nivel. Los *trazadores gráficos (plotters)* sirven para trazar, mediante el desplazamiento de plumas especiales, dibujos o gráficos complejos.

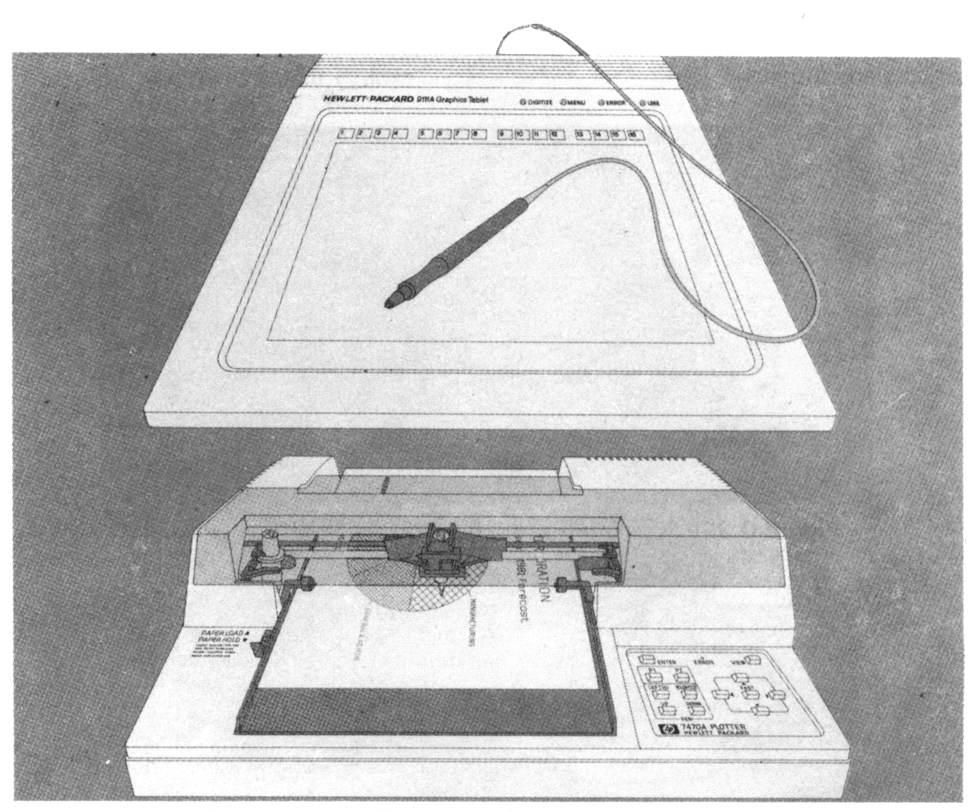


Figura 0.31. Tableta gráfica y trazador gráfico de HP.

9. COMUNICACION ENTRE PERIFERICOS Y COMPUTADORAS. INTERFACES

Para conectar periféricos a la computadora se necesitan cables, jacks y conectores que normalmente están normalizados y por fortuna existen en el comercio. El dispositivo que conecta un periférico con la unidad central se llama **interfaz** (puede ser un concepto de hardware o de software). Un jack técnicamente llamado **puerto,** *puerta* (*«port»,* vía de acceso), permite comunicar a la máquina con el mundo exterior.

Los **interfaces** pueden ser *serie* o *paralelo.* **Serie** significa que la información viaja sobre cada hilo del cable, un bit después del otro. **Paralelo** significa que varios **bits** (normalmente ocho) viajan por hilos separados simultáneamente y a continuación otros ocho. Los interfaces estándares que suelen incorporar la mayoría de las computadoras y sus periféricos son *Serie:* **RS-232;** *Paralelo:* **Centronics** e **IEEE-488.**

La comunicación entre computadoras o terminales con computadoras centrales a distancias próximas o lejanas puede ser mediante un **modem** (modulador-demodulador) o por un **acoplador acústico.** A través de la línea telefónica se pueden comunicar entre sí diferentes computadoras.

Un *modem* es un dispositivo que abre la puerta de la computadora al mundo exterior, permitiendo a dicha computadora comunicarse o «charlar» con otra situada en el mismo edificio o decenas, centenares de kilómetros de la primera. Un modem convierte el lenguaje

de la computadora (códigos digitales) en señales audio o sonido que se pueden transmitir o enviar sobre un par de hilos ordinarios conectados a un teléfono comercial. En el otro extremo, un modem similar cambia las señales de audio de nuevo a códigos digitales para recibir u «oír» a la computadora.

Los modems se pueden conectar directamente a una línea telefónica con un enchufe modular como el que viene en los teléfonos o utilizar un *acoplador acústico* en el que se pone una pieza del teléfono (a veces el acoplador acústico y el modem están integrados en una sola unidad). En algunos sistemas los modems se construyen en la computadora en vez de ser una unidad separada. Una *tarjeta modem* se debe insertar en computadoras con ranuras vacías para tarjetas adicionales.

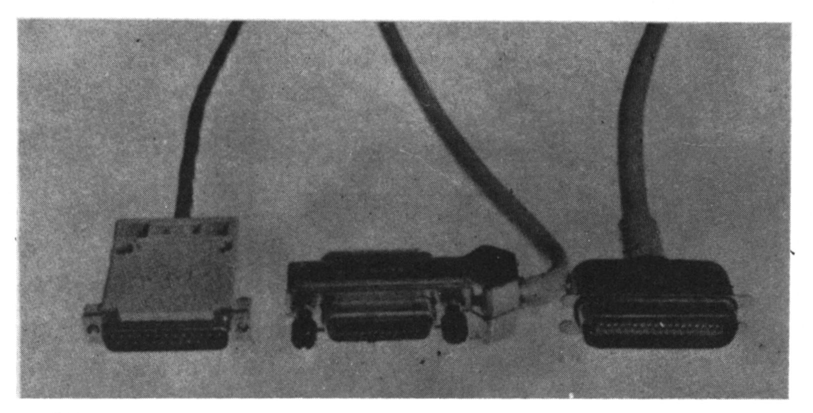


Figura 0.32. Conectores tipo RS-232C, IEEE-488 y Centronics.

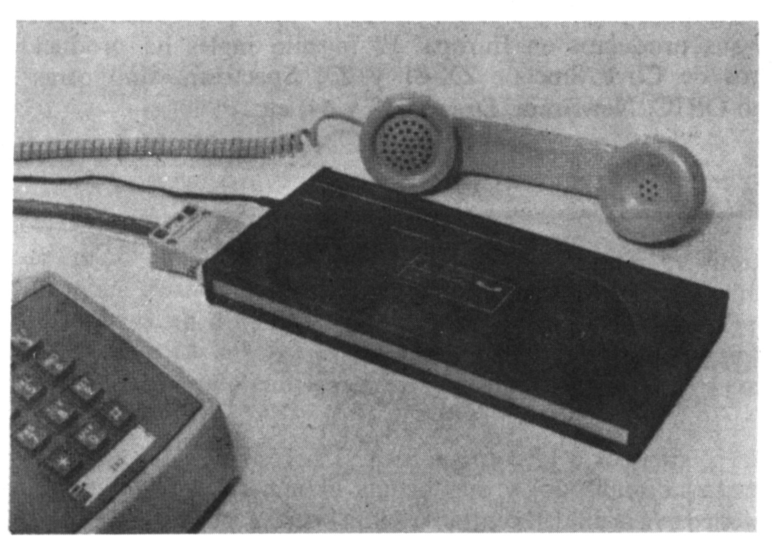


Figura 0.33. Modem.

10. LAS MICROCOMPUTADORAS DE LOS OCHENTA

Cada microcomputadora tiene un tipo diferente de microprocesador o chip utilizado como unidad central. Los microprocesadores y microcomputadoras más utilizadas son:

ROCKWELL 6502 (8 bits)
Apple II, Commodore 64/128, Atari

INTEL 8088, 8086 (16 bits) (IBM PC y compatibles)
IBM/PC, Ericcson PC, Toshiba PC, Sperry PC, ITT Xtra, Digital Rainbow, IBM XT

ZILOG Z-80 (8 bits)
Amstrad PCW8256, Amstrad PCW 8512, Amstrad CPC464/CPC6128, MSX, ZX Spectrum, TRS-80

MOTOROLA 68000 (16 bits)
Macintosh (Apple), Atari 520 ST, Macintosh Plus, Atari 1040 ST, Amiga (Commodore)

MOTOROLA 6809 (8 bits)
Dragón 64/200, TRS-80 Color

INTEL 80286 (IBM AT y compatibles)
IBM AT, HP Vectra, Sperry XP



Figura 0.34. Sistema de computadora ATARI 520ST.

La longitud de palabra de las computadoras domésticas más usual es 8 bits, sin embargo desde los años 82 y 83 han aparecido con una gran fuerza los «micros» de 16 bits como las computadoras personales de IBM, Epson, Sperry, Olivetti, Computer, Bull, etc.

Las ventas de computadoras personales de gestión o profesionales en el año 1984 según los datos de la empresa consultora EICE-IDC fueron los siguientes:

IBM	35 %
Olivetti	7 %
Apple	5 %
Hewlett-Packard	7 %
ACT	4 %
Victor	4 %
Sanyo	3 %
DEC	3 %
Wang	3 %
Otros	29 %

En comparación, en España las ventas totales (en millones de pesetas) de equipos informáticos en el año 1985, según la revista Chip (Marzo 86) fueron:

IBM	191.982
Nixdorf.	17.000
NCR	15.450
Olivetti	14.000
Sperry	14.000
Secoinsa	12.000
Philips	11.800
Bull	11.800
Digital	7.414
Hewlett-Packard	7.100

Los nuevos mercados inglés y japonés presagian para la segunda mitad de esta década la invasión de sus productos en Europa. El ingenio inglés ha producido no sólo los populares micros de Clive Sinclair, ZX-81 y ZX Spectrum, sino otras computadoras personales como Oric, NewBrain, Dragón-32 y 64, y, últimamente, la saga de Amstrad que, con sus modelos CPC-464, CPC-6128, PCW 8256 y PCW 8512, han inundado el mercado europeo, consiguiendo un notable éxito de ventas en España en estos dos últimos años.



Figura 0.35. Computadora Amstrad 8256 diseñada específicamente para tratamiento de textos y con características semiprofesionales para gestión.

11. LENGUAJES DE PROGRAMACION

Una computadora es una máquina que sólo comprende las instrucciones que se le den en un determinado formato y que a semejanza de los humanos se llama *lenguaje de programación.* Cada máquina reconoce y ejecuta un pequeño número de instrucciones (decenas o centenas) diferentes que se agrupan en lenguajes como BASIC, FORTRAN, Pascal, Ensamblador, etc.

Un **lenguaje de programación** es un conjunto limitado de palabras y de símbolos que representan procedimientos, cálculos, decisiones y otras operaciones que puede ejecutar una computadora. Los lenguajes que son parecidos, normalmente al inglés, se llaman *lenguajes de alto nivel* y utilizan un número reducido de instrucciones que siguen unas estrictas reglas gramaticales que se conocen como *sintaxis del lenguaje.* Así pues, *un lenguaje de programación* es un conjunto de reglas conocidas por sintaxis, palabras y símbolos o instrucciones a la computadora.

La máquina trabaja en *lenguaje máquina* (números digitales con sólo cifras 0 y 1). Para que pueda comprender un programa escrito en lenguaje de alto nivel es preciso que disponga internamente de un traductor de lenguaje que lo traduzca a la máquina bajo la forma de una secuencia de instrucciones. Existen dos tipos de programas traductores según el método empleado: **compilador,** traducción global del programa; **intérprete,** traducción inmediata en el momento de la ejecución.

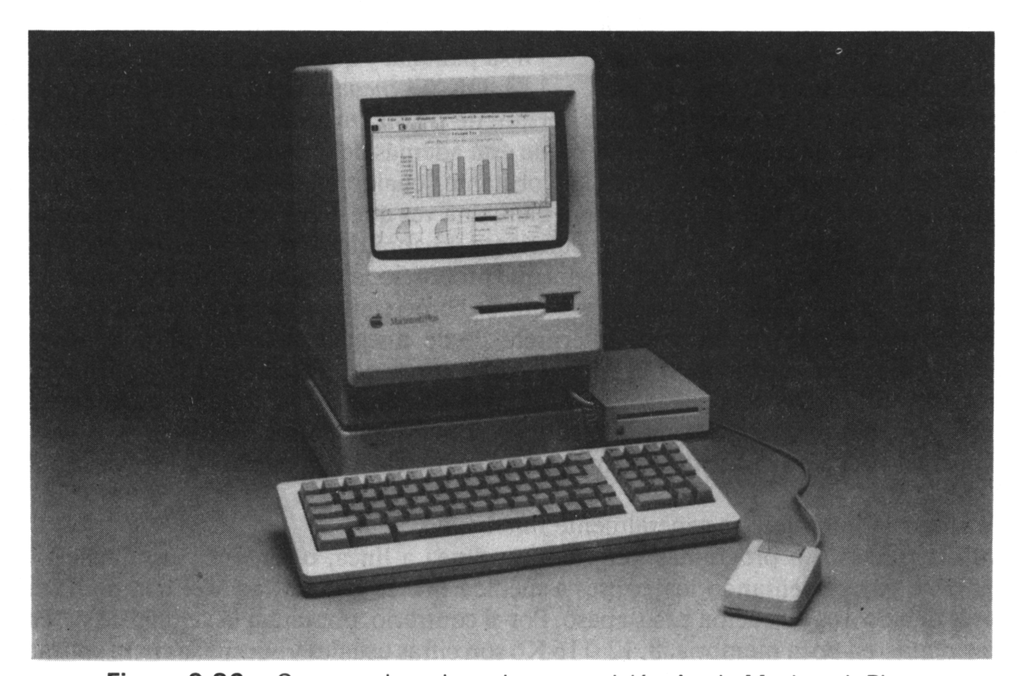


Figura 0.36. Computadora de reciente aparición Apple Macintosh Plus con unidad de disco y ratón.

11.1 El compilador

El compilador es un programa utilizado en lenguajes de alto nivel FORTRAN, COBOL, PASCAL, etc., y en los últimos años BASIC en algunos modelos de computadora. Permite traducir un programa escrito en estos lenguajes a lenguaje máquina. La operación de traducción se llama *compilación* y la operación la realiza un programa llamado compilador.

El compilador traduce el programa completo antes de que éste se ejecute, por lo que la ejecución se realiza en un período muy breve. Como referencia se puede afirmar que, en general, un compilador ejecuta un programa a una velocidad diez veces mayor que un intérprete.

La gran ventaja del compilador reside en la ejecución de todas las instrucciones o sentencias a la vez, mientras que el intérprete las ejecutará una a una. Un ejemplo ilustrativo son las estructuras de tipo cíclico o bucles en las que unas determinadas instrucciones se repiten un

número determinado de veces; el compilador las ejecuta simultáneamente y el intérprete una a una. Sin embargo, frente a esta ventaja el compilador presenta el inconveniente de necesitar más memoria y ser más complicado su manejo.

En la actualidad, como ya se ha dicho, comienzan a aparecer compiladores BASIC, lo que les proporcionará a esas versiones una gran potencia y velocidad de cálculo. Sin embargo, todavía las versiones de lenguajes BASIC más frecuentes son intérpretes.

11.2. El intérprete

El **intérprete**, como su nombre indica, es un programa interno de computadora que interpreta los símbolos del programa y los traduce a lenguaje máquina. Un intérprete BASIC lee cada instrucción escrita mediante teclado en lenguaje BASIC y la traduce automáticamente en una secuencia de instrucciones que la computadora comprende y ejecuta. Este proceso exige en la *jerga* informática que la computadora sepa *hablar en BASIC* o lo que es igual que el programa intérprete esté alojado en su memoria. En función de esta ubicación los programas intérpretes pueden ser **residentes o no residentes**, siendo los más utilizados en **microcomputadoras** y **computadoras personales** los intérpretes residentes. El intérprete se llama *residente* cuando reside permanentemente en la memoria de la computadora y está disponible desde que se enciende el mismo. El modo de saber cuando está la computadora preparada para la ejecución de instrucciones o programas en BASIC es la aparición de un carácter orientativo, de guiado o de petición (*prompt* en inglés) que aparece en su pantalla y que puede variar de una computadora a otra, siendo los más frecuentes: ?, >, ▨, OK. El *inconveniente* mayor de un intérprete residente es que puede proporcionar una versión de BASIC *mínima,* debido a la limitación de la memoria central de la computadora y a que ésta además del programa intérprete tendrá que contener otros programas, datos, etc. No obstante, cualquier BASIC reciente le permitirá aprender y trabajar con este lenguaje; ahora bien, una vez que desee realizar programas complejos seguramente su intérprete no le será suficiente y habrá de recurrir a uno *no residente.*

El intérprete *no residente* es un programa que proporciona versiones de BASIC más potentes conocidas como *extendidas* o *avanzadas* y que se comercializan en casete o disquete. Para su utilización será preciso tranferir el intérprete desde disquete a la memoria central de la computadora; a esta operación se la conoce como *carga* del intérprete y se realiza mediante instrucciones u órdenes adecuadas. El BASIC está disponible cuando aparece el carácter de orientación, que normalmente será diferente en el intérprete no residente.

Cuando el BASIC es residente, el tamaño de la memoria ROM puede dar idea de la potencia del BASIC.

Un lenguaje intérprete es generalmente *interactivo.*

Los intérpretes son programas que traducen línea a línea, o sentencia a sentencia, y por consiguiente van presentando los errores a medida que se encuentran y el tiempo de cálculo es grande debido a su ejecución paso a paso. Por el contrario, presentan la ventaja de su sencillez de uso y requieren poca memoria. 8, 12 o 16 Ko son cifras usuales y adecuadas para aplicaciones *conversacionales* al permitir un diálogo interactivo usuario/máquina. Para evitar los problemas anteriores existen los programas compiladores.

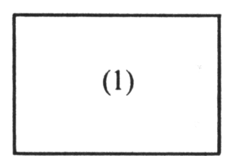
Los lenguajes intérpretes permiten desarrollar y modificar un programa de modo interactivo, lo que facilita la puesta a punto de los programas. El precio es una ejecución más lenta y a veces programas más difíciles de ejecutar o estructurar.

12. DIAGRAMAS DE FLUJO

Un diagrama de flujo es una representación gráfica del flujo lógico de datos que se utilizarán en la formulación de un programa. El diagrama de flujo se dibuja *antes* de que el programa se escriba para asegurar que las instrucciones se integrarán lógicamente. Antes de la codificación del programa es conveniente que los usuarios revisen los diagramas de flujo con los programadores para asegurar que éstos comprenden completamente los requisitos de la tarea a realizar.

Una herramienta llamada **plantilla** se utiliza para dibujar los símbolos estándares en un diagrama de flujo. Los símbolos más utilizados y la funcón que realizan son los siguientes:

Símbolo — **Función o significado**



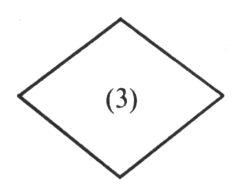
1. PROCESO

Este símbolo indica cualquier proceso interno de la computadora, esto es, cualquier serie de transferencia de datos u operaciones aritméticas.

(2)

2. ENTRADA/SALIDA — E/S

Este símbolo representa cualquier operación de entrada o salida, como LECTURA, ESCRITURA, etc. La operación específica se realiza indicándola por medio de un comentario dentro del símbolo.



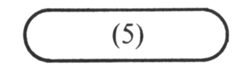
3. DECISION

Este símbolo se utiliza para verificar una comparación lógica. Básicamente se utiliza cuando la computadora ha de realizar una pregunta.



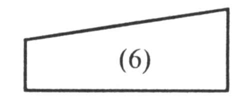
4. CONECTOR

Este símbolo significa un punto de referencia que indica donde debe continuar el diagrama de flujo. Se utiliza para indicar un cambio en el flujo normal de datos (transferencia o bifurcación).



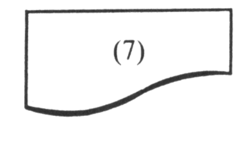
5. TERMINAL

Este símbolo significa el comienzo o punto final de un programa.



6. ENTRADA DESDE TECLADO

Introducción de datos desde el teclado.



7. SALIDA HACIA UNA IMPRESORA

Presentación de resultados en impresora.

8. SENTIDO FLUJO DE DATOS

Conexión lógica entre unos símbolos y otros.

REGLAS PARA LA REALIZACION DE DIAGRAMAS DE FLUJO

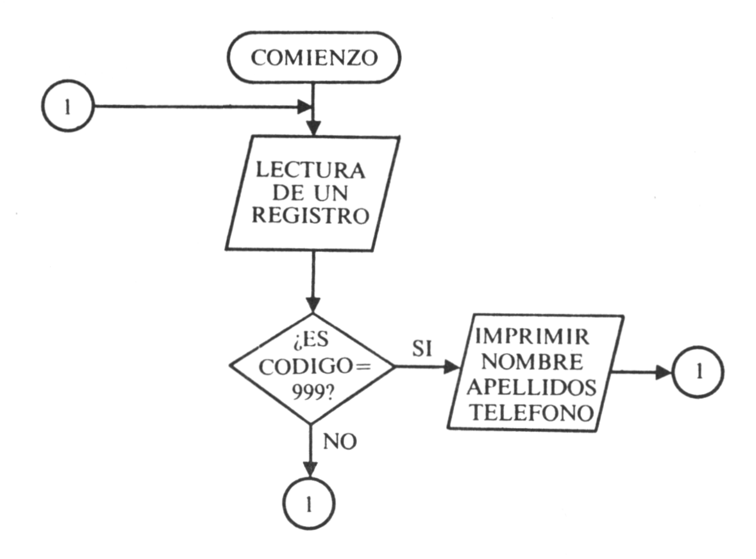
1. Cada símbolo significa un tipo de operación:

    Entrada/Salida
    Proceso
    Decisión
    Transferencia o bifurcación

2. Un comentario se escribe dentro de cada símbolo para indicar la función específica que se ha de ejecutar.
3. Los diagramas de flujo se leen de arriba a abajo.
4. Una secuencia de operaciones se ejecuta hasta que un símbolo terminal designa el final de la ejecución o un conector de *bifurcación* transfiere el control.

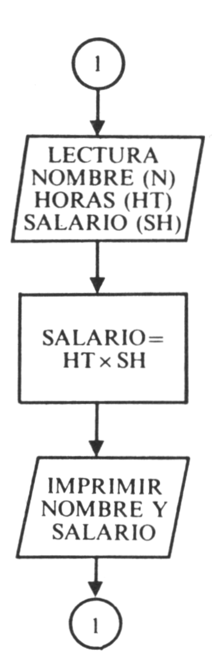
Ejemplo 1

El diagrama de flujo siguiente indica la lógica para la impresión del nombre, apellidos y número de teléfono de cada persona anotada en un registro (ficha) de un fichero. La figura describe la salida que se prodcirá a partir del programa codificado de este diagrama.



Ejemplo 2

Imprimir el nombre y el salario semanal para cada empleado de una ficha (registro) de una cinta, en donde se lee el nombre (N), horas trabajadas (HT) y salario hora (SH). El diagrama de flujo se muestra a continuación:



Ejemplo 3

Utilizando el problema del ejemplo anterior, imprimir al final de la ejecución del programa el número total de registros (fichas) procesados además del nombre y salarios.

a) Determinación del final de la ejecución.

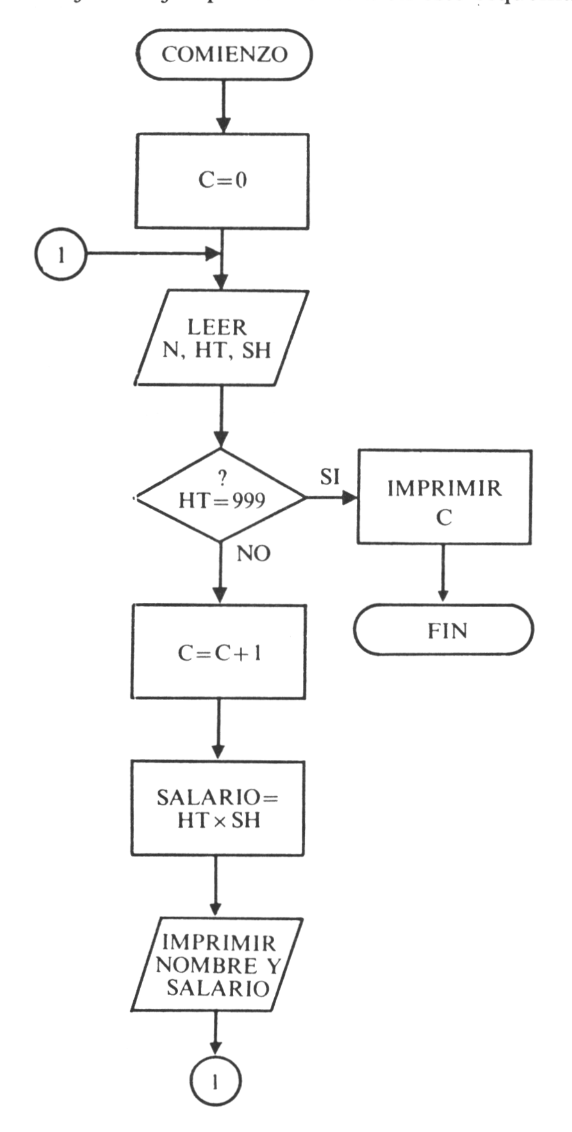
En una cita o disco hay un indicaro de fin de fichero. En ausencia de tal indicador, un registro falso se utiliza para señalar un fin de fichero. Por consiguiente, podemos designar arbitrariamente un fin de fichero como un registro con 999 en HT, ya que lógicamente ninguno trabajará 999 horas en una semana; 999 significará que se desea bifurcar al final de una subrutina «fin de fichero» donde se imprime el número total de registros.

b) ¿Cómo contar el número de registros?

Lea por ejemplo un campo de la ficha llamado C que servirá como nuestro contador. C debe comenzar en 0. Cada vez que un registro se lee, se suma 1 a C. El símbolo de proceso puede ser

ASIGNAR 1+C a C	o	C=C+1

El símbolo último significa poner en C el valor último C+1, o lo que es igual, incrementar C en 1. El diagrama de flujo del ejemplo se muestra en este esquema:



Ejemplo 4

Supongamos se desea calcular la suma de los N primeros números enteros.

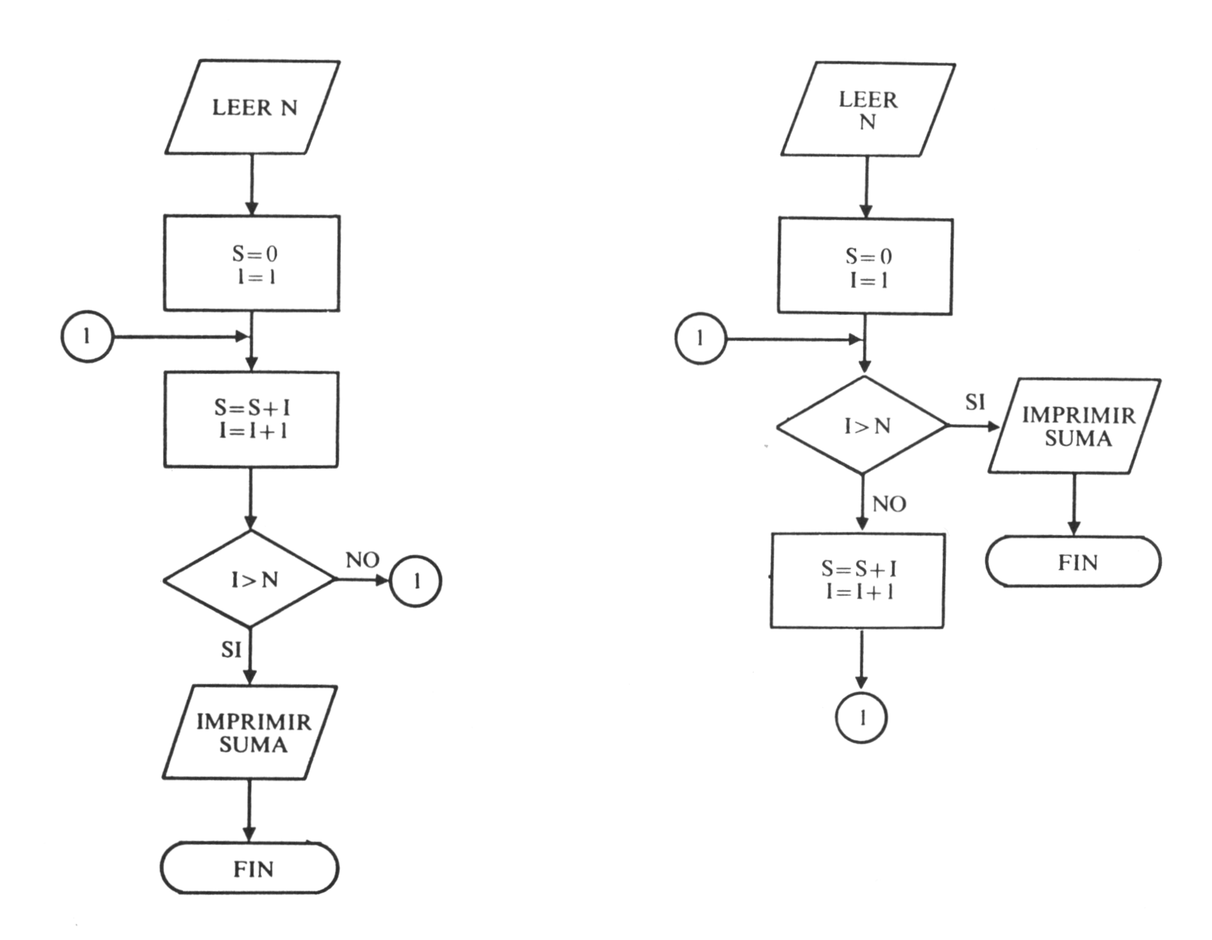
El algoritmo puede ser especificado por una fórmula de recurrencia. Suponiendo conocida la suma de los I-1 primeros números, S_{I-1}, se obtiene la suma de los I primeros números añadiendo I a S_{I-1}.

S I=S_{I-1}+1 en el diagrama de flujo S=S+I

El proceso sería el siguiente:

Paso 1: Leer N.
Paso 2: Inicializar la suma y en contador de enteros: 2=0, I=1
Paso 3: Sumas parciales: S=S+1.
Paso 4: Incremento del contador: I=I+1.
Paso 5: Decisión: Si I > N imprimir suma, S y terminar (FIN). Si I ≤ N volver al paso 3.

El diagrama de flujo podría ser cualquiera de los dos que siguen:



CAPITULO 1

Introducción a la programación

0. INTRODUCCION

A lo largo de este capítulo aprenderá a:

— Manejar la computadora, en modo directo, como si fuese una calculadora.
— Ejecutar, modificar y depurar un programa.
— Editar un programa correctamente haciendo uso de las órdenes (comandos) de edición más frecuentes.
— Cargar, borrar y conservar o grabar programas.
— El alfabeto del lenguaje BASIC.

1. GENERALIDADES

El lenguaje BASIC es un lenguaje de programación de alto nivel, muy especializado y diseñado para permitir el diálogo interactivo entre la computadora y el usuario, no es complicado y es mucho más fácil de aprender que un lenguaje hablado tal como el inglés o el francés.

El BASIC, al haberse hecho uno de los lenguajes de programación más difundidos, ha dado lugar a una proliferación de versiones muy grande, de modo que aunque todas ellas conservan analogías en cuanto a las sentencias fundamentales, las distintas versiones presentan diferencias que a veces son tan notables que obligan al usuario de cada computadora, aunque sea un experto programador, a la lectura detenida de los manuales de programación del fabricante, a fin de obtener el máximo rendimiento de su computadora.

Debido a lo anterior, se irán recogiendo a lo largo de este libro las versiones BASIC de mayor difusión: MICROSOFT (MBASIC, MSX BASIC, BASIC 80, BASICA del IBM, GW BASIC, etc.), APPLESOFT, CBM, HP BASIC, CBASIC, ANSI, MSX, Amstrad, etcétera, de modo que este curso de BASIC pueda servir a los usuarios actuales o futuros de las computadoras más populares que trabajan en este lenguaje: IBM/PC, APPLE, Commodore, SINCLAIR, TRS-80, ATARI, HEWLETT PACKARD, DRAGON, Amstrad, MSX, etc.

BASIC tiene un vocabulario sencillo (por desgracia en idioma inglés) que consta de unas pocas palabras, una estructura gramatical y reglas de uso como cualquier otro lenguaje. Su principal tarea será aprender el vocabulario de BASIC hasta que se familiarice con sus reglas de gramática, y comience a ver cómo el lenguaje le permite utilizar la computadora para hacer cuanto desee (dentro de unos límites , ¡claro!). El nivel de matemáticas exigido es muy elemental para programación de tipo normal. Sólo en el caso de cálculos técnicos sofisticados le será necesario conocimientos matemáticos profundos.

Un camino muy efectivo para aprender es observar detalles y características mientras se ejecuta una tarea: el método de descubrimiento. Esta táctica se utilizará con frecuencia a lo largo del curso. Si dispone de una computadora deberá introducir y ejecutar todos los progra-

mas ejemplos que se le vayan mostrando, así como todos los que se le propongan como ejercicio.

La computadora realiza su misión mediante la ejecución de un programa. Así pues, ¿qué es un programa?

Un programa es una serie de instrucciones a ejecutar, realizadas mediante frases escritas en un lenguaje definido por un alfabeto y un conjunto de reglas de formación de estas frases. Las reglas constituyen la *sintaxis o la gramática* del lenguaje.

El campo de aplicación del lenguaje BASIC es muy completo y en la actualidad se le utiliza desde la resolución de problemas científicos al tratamiento de problemas de gestión, pasando por una amplia gama de aplicaciones como: enseñanza, tratamiento de textos, estadísticas, etc. Asimismo el tratamiento de BASES de DATOS ya es realizable por las microcomputadoras.

Por todas estas razones hemos afrontado la idea de realizar un curso completo de BASIC y presentar en él las versiones actuales de los diferentes tipos de microcomputadoras.

Como ya se ha avanzado antes, una de las grandes ventajas del lenguaje BASIC es ser un lenguaje de programación *«interactivo» o «conversacional».*

¿DONDE SE ORIGINO EL BASIC?

La versión original de BASIC se diseñó y escribió en Darmouth College bajo la dirección de G. Kemeny y Thomas E. Kurtz*. En 1963 comenzaron a trabajar en un proyecto para crear un lenguaje de programación escrito desde el punto de vista del usuario. El nacimiento del BASIC se suele considerar en mayo de 1964.

Kemeny y Kurtz han diseñado una nueva versión de BASIC denominada True Basic que promete ser una revolución en el mundo de los lenguajes de programación, como supuso su primera versión, debido a utilizar técnicas interactivas y a la vez técnicas de programación, estructuradas de otros lenguajes de alto nivel como Pascal. Sólo el tiempo podrá confirmar las previsiones.

Las sentencias y órdenes del lenguaje se componen de palabras inglesas, aunque evidentemente no es necesario conocer este idioma para trabajar con microcomputadoras. Bastará simplemente que se aprenda las palabras claves, muy sencillas por otra parte, para obtener el máximo rendimiento en su programación. En realidad, con 10 o 12 palabras diferentes se pueden escribir una gran cantidad de programas.

BASIC es un lenguaje interactivo e intérprete, es decir, cada instrucción se traduce a lenguaje máquina y se ejecuta inmediatamente cuando se reconoce como correcta. Si hay errores se presentan de modo inmediato y se pueden corregir.

Se podrían resumir las **ventajas** del BASIC en las siguientes:

1) BASIC está *orientado al usuario.* Es fácil de aprender y de utilizar. No se requieren conocimientos matemáticos importantes y sólo un poco de sentido lógico.

2) BASIC es *flexible,* permite modificar programas con el mínimo esfuerzo. La depuración o corrección de programas es notablemente fácil.

3) BASIC se puede considerar *el lenguaje de programación más utilizado.*

4) BASIC es *independiente de la computadora de trabajo,* excepto las diferencias ya comentadas. Cualquier persona que conozca el lenguaje BASIC de una microcomputadora tardará unas horas o, a lo sumo, unos días en familiarizarse con el BASIC de otra microcomputadora.

Entre los **inconvenientes** se podrían citar:

1) Es un lenguaje de alto nivel del tipo intérprete y, en consecuencia, cada instrucción se traduce a lenguaje máquina y se ejecuta de modo inmediato tan pronto como se reconoce correctamente. Esta característica lo hace más lento que los lenguajes compiladores. Los lenguajes FORTRAN y Pascal tienen ventaja en este aspecto al ser compiladores.

2) La diversidad de versiones BASIC, aunque realmente se podría considerar como incomodidad de trabajo y no como inconveniente.

3) Es un lenguaje no estructurado.

---

* En el año 1985 Kurtz y Kemeny han diseñado una nueva versión de BASIC conocida como True BASIC, que incorpora estructuras tipo Pascal y que promete ser un éxito en el año 1986 y siguientes.

Para remediar la lentitud del intérprete BASIC y su «no estructuración», están apareciendo con gran profusión versiones de lenguajes BASIC compiladores como CBASIC, y versiones estructuradas como es el caso del BASIC de las computadoras QL, BBC, Macintosh, HP, etc.

2. BASIC EN MODO DIRECTO O INMEDIATO

La computadora se encuentra en *modo directo o inmediato* nada más encenderla. Este modo de trabajo es similar al de una calculadora.

El modo inmediato se suele indicar mediante un indicador o cursor en la pantalla, normalmente parpadeante,], >, ▨, –, etc., y mensajes tales como **Ready, OK,** etc. A lo largo del curso se utilizará la palabra OK, lo que nos indicará que la computadora está preparada para recibir órdenes en modo directo o en modo programa, como se verá más tarde.

En modo directo la computadora ejecutará inmediatamente las sentencias BASIC que se le dan. Interpreta dichas sentencias mediante las reglas del lenguaje BASIC, nada más introducir los mensajes correspondientes. Esta tecla suele llamarse **RETURN,** aunque hay computadoras que utilizan **NEW LINE, ENTER, STORE** o **CR**. A lo largo del curso se empleará **RETURN**, dado que suele ser la más utilizada.

Aunque más adelante volveremos a hablar de nuevo y con más intensidad sobre la palabra sentencia, vamos a definirla ahora. Una *sentencia o instrucción* es un conjunto de palabras, separadores de palabras y palabras claves, que constituyen *frases* del lenguaje y que serán las encargadas de instruir a la computadora sobre la tarea que deberá realizar. Sin embargo, todas las frases que se puedan construir no son correctas. Su construcción obedece a ciertas reglas que se denominan reglas de sintaxis y que constituyen *la gramática* del lenguaje.

Veamos cómo trabaja la computadora en modo inmediato, mediante la introducción de la correspondiente sentencia.

Escriba lo siguiente:

`5+6`	*Pulse*	<RETURN>.	*Se visualizará en la pantalla 11*
`9*8`	*Pulse*	<RETURN>.	*Se visualizará en pantalla 72* *(* es el símbolo que representa la multiplicación)*
`2↑8`	*Pulse*	<RETURN>.	*Se visualizará en pantalla 256* *(↑ es el símbolo de la potenciación,* 2^8*)*
`SQR (144)`	*Pulse*	<RETURN>.	*Se visualizará en pantalla 12*

Como se habrá visto, en cuanto se ha pulsado la tecla RETURN la sentencia se ha ejecutado inmediatamente, apareciendo tras el resultado el mensaje OK o bien Ready.

El empleo más corriente del modo directo es para la realización de operaciones aritméticas mediante los operadores correspondientes. Sin embargo, existen algunas sentencias que pueden ser realizadas mediante el modo directo y que como vamos a ver a continuación, amplían la utilidad del modo directo.

Recuerde: Para que una orden se transmita y ejecute por la computadora será preciso que a la terminación de la/s sentencia/s correspondientes se pulse la tecla RETURN o su equivalente. Siempre que vea escrito <RETURN> le estamos indicando que debe pulsar la tecla RETURN o su equivalente.

CORRECCION DE ESCRITURA

Antes de proseguir le vamos a explicar cómo corregir errores de escritura por un procedimiento sencillo, similar al utilizado en las máquinas de escribir manuales; posteriormente le describiremos un procedimiento más completo.

La corrección de caracteres normalmente mal escritos se realiza con la tecla RETROCESO (BACKSPACE) situada normalmente en la esquina superior derecha de la pantalla y que suele representarse por **BACKSPACE** o bien por ←.

Esta tecla hace retroceder el cursor un espacio cada vez que se pulsa y además borra el carácter justo encima del cursor (si no desea borrar el carácter y sólo retroceder un espacio, utilice la tecla de movimiento de cursor hacia la izquierda).

Escriba la frase:

```
Hola Mckoa
```

Pulse la tecla RETROCESO (BACKSPACE o bien ←) y tras borrar el carácter **a,** pulse la tecla **y.** La frase quedará como sigue:

```
Hola Mckoy
```

2.1. Sentencia PRINT

La sentencia PRINT permite imprimir caracteres en la pantalla o en la impresora, bien como mensajes o como resultado de operaciones.

Esta sentencia es la más usada en modo directo y da instrucciones a la computadora para visualizar datos numéricos, alfanuméricos (letras, números y símbolos de escritura) o caracteres especiales o gráficos.

El *formato* de la sentencia es:

PRINT *expresión numérica, alfanumérica o variables.*

Ejemplos: Escriba las siguientes sentencias pulsando al terminar RETURN.

```
PRINT "BUENOS DIAS"              <RETURN>
BUENOS DIAS
PRINT "3 DE FEBRERO DE 1983"     <RETURN
3 DE FEBRERO DE 1983
PRINT 10+6                       <RETURN>
 16
PRINT 500-250                    <RETURN>
 250
PRINT (3+4) *2                   <RETURN>
 14
PRINT 3+3/2-1                    <RETURN>
 3.5
PRINT 3^2                        <RETURN>
 9
```

Como se ve, la respuesta en la pantalla va a continuación en la línea siguiente a cada sentencia.

La expresión de frases (cadenas de caracteres) exige que éstas se encuentren entre comillas, mientras que los datos numéricos no tienen esta limitación.

Atención: Si al pulsar RETURN tras escribir alguna sentencia se imprime el mensaje ?SYNTAX ERROR, *** SYNTAX ERROR, o alguno similar, significa que su sentencia es indescifrable o, lo que es igual: se escribió mal. Repase su instrucción, probablemente deletreó mal o se olvidó de las comillas, si tenía que ponerlas. Si se olvidó de las primeras comillas, se imprimirá como resultado 0 o un mensaje de error según el tipo de computadora.

```
PRIMT 45*3              <RETURN>
?SYNTAX ERROR           Escribió una M en lugar de N
PRINT BUENOS DIAS"      <RETURN>
 0                      Se olvidó abrir las comillas
```

Sin embargo,

```
PRINT "BUENOS DIAS"  <RETURN>
BUENOS DIAS
```

El número máximo de caracteres que se pueden escribir en una sola instrucción suele ser de 255 caracteres o bien dos líneas completas de pantalla (80 caracteres en la pantalla de 40 columnas).

NOTA: *Muchas computadoras admiten como abreviatura de la palabra* PRINT *el signo de interrogación, ?, con lo que se reduce el tiempo de escritura de los programas, sobre todo si constan de un gran número de sentencias* PRINT.

Así:

```
?"BUENOS DIAS"      Equivale a  PRINT "BUENOS DIAS"
? 3*4               Equivale a  PRINT  3*4
```

2.2. Sentencia LET

Para observar en memoria un valor o el resultado de un cálculo se le asigna un nombre mediante la sentencia LET (asignación). También se le asigna un nombre acompañado del símbolo $ a las cadenas de caracteres. A estos nombres se les conoce por *variables.*

```
LET A=5               El valor 5 se almacena en la posición de memoria llamada A.
LET B=4.32            El valor 4.32 se asigna a la posición B.
LET A$="BUENOS DIAS"  La frase BUENOS DIAS se almacena en la posición de memoria A$.
```

Posteriormente, estos valores asignados podrán ser recuperados para cálculos, presentaciones o impresiones.

Introduzca las sentencias siguientes pulsando como siempre a continuación la tecla RETURN.

```
LET A=5.9    <RETURN>
LET B=0.6    <RETURN>
LET C=A-B    <RETURN>
PRINT C      <RETURN>
```

En la pantalla se visualizará:

```
5.3
```

ESPACIOS EN BLANCO

En BASIC los espacios o lugares en blanco carecen de importancia, ya que no son significativos y la computadora no los interpreta. Sólo tienen importancia cuando van en frases o números entre comillas dado que en este caso, lógicamente, se representarán así en pantalla.

Así:

```
LETA=5.2    Se interpreta igual que    LET A=5.2
```

Unicamente con fines de legibilidad se suelen dejar espacios en blanco, pero no se preocupe si escribió una sentencia y a continuación una variable sin dejar espacio en blanco, su computadora la interpretará correctamente.

Por tanto, en el único sitio donde tiene que poner obligatoriamente espacios en blanco es en las frases entre comillas, donde quiera que el espacio forme parte del lenguaje.

NOTA: *La mayoría de las versiones* BASIC *que existen hoy día admiten, como sentencia de asignación* LET, *la igualdad lógica* (=) *suprimiendo la palabra* LET.

Así:

```
LET B=5    es equivalente a    B=5
```

De esta forma lo usual es trabajar sin la palabra LET.

```
A=5            <RETURN>
B=6            <RETURN>
C=7            <RETURN>
PRINT A+B+C    <RETURN>
 18
```

LINEAS MULTISENTENCIA/PROGRAMAS DE UNA LINEA

La mayoría de las versiones de BASIC admiten más de una sentencia en una sola línea, con tal de que estén separadas mediante un carácter separador. De esta forma, aunque en modo directo, los programas sólo tienen una línea, sin embargo podrán realizarse algunas tareas simples utilizando este formato.

Los separadores más utilizados en computadoras son:

: COMMODORE, IBM PC, MSX, Amstrad, APPLE II, DRAGON-32, TRS-80, etc.
@ HEWLETT PACKARD (HP-85, HP-86, HP-87, etc.)

Introduzca y ejecute la siguiente línea (este conjunto de sentencias forman lo que se denomina *programa* de una línea).

```
A=5: B=6: PRINT A*B: PRINT A+B   <RETURN>
 30
 11
```

Aunque aún no conozca el significado de la sentencia FOR-NEXT, introduzca por teclado la siguiente línea y observe el resultado de su ejecución, después de pulsar RETURN.

```
FOR N=1 TO 5: PRINT "*": NEXT N
*
*
*
*
*
```

En este libro se utilizará el separador (:) por considerar que es el más extendido.

3. BASIC EN MODO PROGRAMA

El modo programa o programado es el modo normal de funcionamiento en BASIC, dado que los inconvenientes del modo inmediato no lo hacen útil para aplicaciones de ligera complejidad. Recordemos que las limitaciones más sobresalientes del modo inmediato son: 1) limitación del programa a una sola línea, y 2) se ha de teclear el programa cada vez que se quiere utilizar.

La forma de evitar los inconvenientes del modo inmediato es escribir los programas en *modo programado,* llamado a veces *indirecto o diferido.* En este modo, la computadora acepta y almacena el programa en su memoria, pero no realiza ninguna instrucción hasta que así se le indique de forma expresa.

Un programa consta de una o varias líneas numeradas que instruyen a la computadora de un modo exacto y completo de la tarea a realizar; cada línea consta de una o varias sentencias BASIC que cuando se ejecutan en la sentencia adecuada realizan la tarea encomendada o misión solicitada. Se pueden introducir tantas líneas como se desee.

El programa visto anteriormente era trivial, sólo tenía una idea; la mayoría de los programas tienen decenas, centenas e incluso millares de líneas.

Los programas se pueden introducir en la computadora directamente por el usuario desde el teclado o cargarse en la memoria de la computadora desde un periférico externo: casete o unidad de disco (flexible o rígido).

Si el programa se introduce desde el teclado, cada línea del mismo se escribe con un número indicativo de la misma. Al final de la línea se pulsa la tecla RETURN (o su equivalente), con lo que la/s sentencia/s de la misma se almacenan en la memoria hasta que sean usadas posteriormente. Una vez que se ha introducido todo el programa, se está en condiciones de ejecución del mismo. Todas las sentencias almacenadas permanecen en memoria hasta que se borra el programa o hasta que se apaga o desconecta la computadora. Por ello, para evitar la pérdida del programa en memoria se debe conservar en una cinta casete, disquete, unidad del disco rígido o Winchester. Esta conservación evita las pérdidas fortuitas producidas por «cortes de luz» imprevistos, así como el uso posterior y a voluntad del programa sin necesidad de escribirlo cada vez que se desee ejecutar. Estando el programa en un medio externo (casete, disquete, etc.) se puede cargar en la memoria de la computadora cuando se desee.

3.1. Ejecución de un programa

Se dice que la computadora *ejecuta* un programa cuando realiza las operaciones o instrucciones que el mismo indica.

Un programa en modo inmediato se realiza tan pronto como se pulsa la tecla RETURN. En modo programa es preciso ejecutar la orden RUN (o teclear la correspondiente tecla RUN, en aquellas computadoras que la poseen en su teclado) para iniciar el programa, de modo que cada vez que se realiza RUN el programa se ejecuta.

El procedimiento de ejecutar el programa es teclear R-U-N (tras haber introducido el programa en memoria, bien por haberlo tecleado, bien por cargarlo de un disco o casete) y pulsar la tecla RETURN (las computadoras potentes suelen incluir una tecla denominada RUN, con la que se ejecuta el programa nada más pulsarla).

Ordenes (comandos) son mandatos directos que se dan a la computadora y que se ejecutan en modo directo. (Al igual que las sentencias, siempre será preciso pulsar la tecla RETURN si la orden se escribió mediante teclado, no siendo necesario si la misma está escrita en una tecla y se realiza al pulsar ésta.)

En la terminología o «jerga» informática la palabra *ejecución* se suele sustituir a veces por *«correr»*, *«lanzar»* o *«rodar»* el programa (*«run»* en inglés).

3.2. Borrado de programas antiguos

Ya que los programas se almacenan en la memoria, cada vez que vaya a introducir un nuevo programa deberá indicarle a su computadora que borre el programa que en este momento hubiese en memoria, antes de teclear el nuevo.

La orden que realiza esta misión es NEW, seguido por la pulsación de RETURN.

Si olvidó teclear NEW <RETURN>, el programa nuevo se mezclará con el antiguo y ello le originará grandes problemas.

Precaución: Dado que la orden NEW borra el programa en memoria, sólo deberá ejecutarlo cuando esté plenamente seguro de que ese programa no le interesa o cuando ya ha sido conservado o grabado en un medio externo (casete, disquete, etc.).

Nota: En algunas computadoras existe una orden SCRATCH que realiza igual misión que NEW.

3.3. Detención o interrupciones de un programa

Un programa se puede *detener* o *interrumpir* durante su ejecución. A esta acción se la conoce como «interrupción» o «rotura» del programa. La orden que realiza esta misión es STOP, que suele venir en casi todas las computadoras en una tecla llamada BREAK o STOP.

Al pulsar STOP se detiene el programa y se visualiza en la pantalla un mensaje de interrupción que indica el número de línea donde se ha detenido.

```
BREAK IN 100
```

La computadora APPLE II no tiene tecla STOP ni BREAK, empleándose para este fin las teclas CTRL y C. Las computadoras IBM PC y compatible ejecutan la orden BREAK pulsando simultáneamente las teclas CTRL y BREAK. Las computadoras Amstrad interrumpen temporalmente el programa al pulsar la tecla ESC y definitivamente al pulsar dos veces la tecla ESC. Las computadoras MSX efectúan la interrupción con las teclas CTRL y STOP.

FIN DE UN PROGRAMA

Mediante la sentencia END, que será la última que se ejecute en un programa, se termina el mismo. END instruye a la computadora para detener la ejecución y vuelve al modo inmediato u órdenes BASIC (representado por el mensaje «Ok»). Para desarrollar buenos hábitos y evitar problemas innecesarios en el futuro, debe finalizar todos los programas con sentencias END.

Sin embargo, hay versiones de BASIC que no necesitan la sentencia END para terminar un programa. Este finaliza cuando se termina de ejecutar la última línea o instrucción. Es el caso de Applesoft en Apple II, CBM BASIC, COMMODORE 64/128, BASICA en IBM PC, etcétera.

Como buena norma de programación, le recomendamos termine siempre sus programas con la sentencia END.

3.4. Líneas de programa

Las sentencias se estructuran dentro de líneas de programa, de modo que cada línea del mismo tiene asignado un único número de línea que es sencillamente un número de uno a cinco dígitos al principio de la línea.

La diferencia entre una línea en modo programa y otra en inmediato es precisamente el número de línea, de modo que la computadora supone que se está trabajando en modo inmediato si la línea no comienza con un número.

Observemos, sin embargo, que la mayoría de las órdenes sólo se utilizan en modo inmediato y por ello no exigen número de línea para su ejecución. Así sucede con RUN, NEW, CONT, etc.

Los números de línea pueden llegar en su numeración hasta un límite que depende de la computadora: Interger BASIC (APPLE II), 0 a 32.767; Applesoft (APPLE II), 0 a 63.999; Commodore 64, 0 a 63.999; IBM PC, 0 a 65.535; Spectrum, 0 a 9.999; MSX, 0 a 65.529, etc.

No obstante, es práctica común numerar las filas de 10 en 10, 100 en 100, etc. Su razón se expondrá más adelante.

El número de líneas máximo de un programa depende de la computadora y de su memoria; por ejemplo, la TRS-80 admite 2.048 líneas.

El número de línea debe ser *único*. No puede haber dos líneas de programas con el mismo número. Si utiliza el mismo número de línea dos veces, la computadora sólo recuerda la última línea.

Es normal introducir las líneas en orden *ascendente,* pero no se preocupe si se olvidó de una línea. Introducida en ese momento, la computadora la clasificará e insertará en su lugar adecuado. Introduzca el siguiente programa en memoria:

```
10 REM DEMO          Números de línea 10, 20, 30, 40, 45, 50
20 A=5
30 B=10
40 PRINT A*B
50 PRINT C
30 B=0               Esta línea 30 sustituye a la anterior línea
45 C=3               30 B=10, y, por tanto, B valdrá ahora 0.
```

NOTA: *A partir de este momento no escribiremos la palabra RETURN, sobreentendiendo que a la terminación de cada línea se deberá pulsar dicha tecla o su equivalente.*

A continuacion *liste* el programa (*listado* de un programa es una presentación en orden ascendente de líneas de todo el programa). Escriba LIST y pulse RETURN. En la pantalla aparecerá:

```
10 REM DEMO
20 A=5
30 B=0
40 PRINT A*B
45 C=3
50 PRINT C
```

Como se ve, la línea 45 se colocó en su lugar pese a ser la última introducida y la línea 30 primitiva B=10 se borró.

La primera línea debe ser el número más pequeño y la última el mayor, pero le insistimos en que si teclea las líneas desordenadas, su computadora las clasificará en orden creciente. Por esta razón, para añadir líneas a un programa que está en memoria puede añadirlas al principio, final o en cualquier parte de un programa, sin más que teclear la línea con el número de línea que le haya asignado.

Es preciso dejar suficiente espacio entre números de líneas para poder añadir nuevas líneas posteriormente.

Existen dos clases de líneas dentro de un programa: líneas físicas y líneas lógicas.

Una *línea física* es una línea en el dispositivo de salida que se está utilizando. Por ejemplo cuando está utilizando el monitor de vídeo, una línea física tendrá 40 u 80 caracteres como máximo de longitud, dependiendo de la anchura que se haya establecido para la pantalla.

Una *línea lógica* (línea de programa), sin embargo, se mide de modo diferente. Puede tener hasta 255 caracteres de longitud y se termina con la pulsación de la tecla RETURN. El lugar donde se pulsa RETURN no es el final de una línea de pantalla sino el final de una línea lógica.

El siguiente ejemplo muestra como una línea BASIC puede constar de más de una línea física.

```
100 PRINT "En un lugar de la Mancha de cuyo nombre no quiero
acordarme, ha mucho que vivia un hidalgo caballero de las
mesnadas temido. "
```

Si prueba a introducir una línea BASIC con más de 255 caracteres, cuando se pulse la tecla RETURN se perderán los caracteres extras.

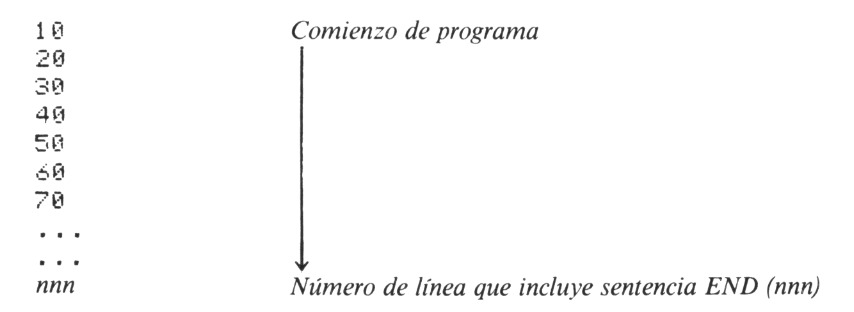
La longitud de cada línea lógica de programa es variable. Así, por ejemplo, en la CBM con pantalla de 40 columnas, una línea de programa admite hasta 80 caracteres de longitud. En la VIC-20 se admiten hasta 4 líneas de pantalla en una línea de programa. Las computadoras con 80 columnas en pantalla admiten en una línea de programa 160 caracteres. La regla práctica suele ser admitir en una línea de pantalla la longitud doble de caracteres admitidos en una línea de pantalla. El Integer BASIC del APPLE II admite 150 caracteres, según el tipo de instrucciones o sentencias. El Applesoft, TRS-80, DRAGON-32, admiten hasta 255 caracteres.

Recordemos que muchas computadoras admiten más de una sentencia por línea de programa, separadas por caracteres tales como (:), @, etc.

Cuando el BASIC ejecuta un programa, trata a las líneas de una en una, comenzando por la de orden inferior y en orden ascendente hasta llegar a la última. Eso significa que los números de línea identifican las posiciones dentro de un programa. Este concepto es importante ya que cada programa contendrá dos tipos de sentencias:

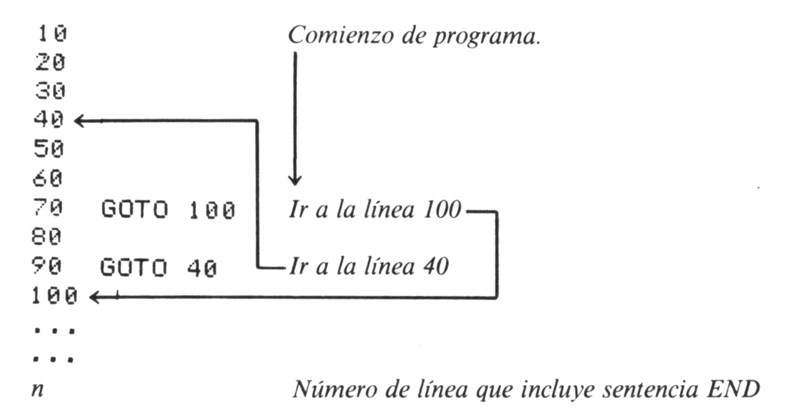
1. Sentencias que crean o modifican datos.
2. Sentencias de control de la secuencia en que se realizan las diferentes operaciones.

La ejecución del programa comienza con la primera línea de programa y continúa secuencialmente del modo siguiente:



Sin embargo, ya veremos que hay sentencias de control condicional como GOTO, IF THEN, GOSUB, etc., que hacen que la secuencia no siga el orden ascendente, sino que mediante saltos cambia la secuencia bien en sentido ascendente o descendente.

Por ejemplo, mediante la sentencia GOTO, que significa *IR A*, la secuencia de un programa podría ser ésta:



Es una buena práctica de programación que proporciona claridad a la lectura del listado dejar entre el último dígito del número de línea y el primer carácter de la sentencia que sigue a continuación uno o más espacios en blanco.

Buena práctica de programación (1)

```
10 A=5
20 B=7
30 PRINT A
40 END
```

Mala práctica de programación (2)

```
10 A=5
20 B=7
30                PRINT A
40 END
```

De hecho hay computadoras que colocan el listado del programa de modo automático tras la introducción de las sentencias por el método (2), en el formato (1), cuando se ejecuta la orden LIST.

Recuerde: Los números de línea.

1. Indican a la computadora que las instrucciones BASIC a continuación del número se almacenan como parte de un *programa* para posterior ejecución (en contraste con las órdenes que son ejecutadas inmediatamente).
2. Definen el *orden* en el que se ejecutarán las sentencias BASIC cuando se ejecute el programa.

3.5. Introducción de un programa en memoria

Una vez escrito el programa sobre una hoja de papel se pasa a introducirlo en la memoria de la computadora. Para ello bastará ir escribiendo cada línea completa con su número de línea y se terminará con la tecla RETURN (ENTER, NEW LINE, STORE o CR, según el tipo de computadora).

Introduzca y ejecute el siguiente programa aunque no entienda lo que hace:

```
5 REM DEMO 1
10 FOR I=1 TO 10
20 PRINT "*";
30 NEXT I
40 PRINT "FIN"
50 END
```

Si durante la fase de introducción (tras la pulsación de RETURN) o al ejecutarlo le aparece el mensaje

```
? SYNTAX ERROR IN LINE N
```

vuelva a escribir e introducir la ínea *n,* ya que habrá cometido un error de sintaxis (más adelante le enseñaremos cómo modificar una línea de programa sin tener que escribirla totalmente).

LIMPIEZA DE LA PANTALLA. Orden CLS

Si la pantalla está llena de líneas de programa, resultados o datos, y desea limpiar o borrar la pantalla completamente, introduzca la orden

```
CLS
```

La pantalla se ha limpiado y el cursor se ha desplazado a su posicón inicial o casa (*home,* en inglés) —esquina superior izquierda de la pantalla. Se puede utilizar la orden CLS siempre que lo desee. Es buena práctica ejecutar CLS cada vez que se ejecuta un programa.

El efecto de CLS se puede conseguir también pulsando la tecla CLS si existe en su teclado, o bien por combinación de teclas: HOME mientras se mantiene oprimida la tecla CTRL, en las computadoras IBM PC y compatibles; las teclas SHIFT y CLS, en las computadoras Amstrad, etc.

CLS no borra el programa que esté almacenando en memoria. Por consiguiente, puede utilizar esta orden sin especiales precauciones. Aunque el efecto conseguido es similar al de NEW, evidentemente la diferencia entre ambos es notable.

ESPACIOS EN BLANCO

Los espacios en líneas BASIC tienen diferentes efectos, dependiendo de donde se presenten en la línea de programa.

En el párrafo 2.2 se vio que su situacióna carecía de importancia en sentencias LET y que, sin embargo, los blancos eran fundamentales entre comillas al señalar una posición de impresión.

Además del caso anterior, en algunas computadoras se requieren espacios en algunos lugares específicos y en otras es recomendable. Por ejemplo, los números de línea y las palabras reservadas o clave BASIC deben de estar separadas por un espacio. En general, las palabras reservadas como PRINT, INPUT, etc. deben estar separadas por un blanco. Una excepción se presenta cuando una palabra clave está precedida por dos puntos (:), ya que en este caso los dos puntos actúan como un espacio.

En los restantes casos, BASIC ignora los espacios extra, a menos que formen parte de una frase entre comillas (cadenas).

En cualquier caso debe tener presente que un espacio en blanco ocupa una posición de memoria (1 byte).

LETRAS MAYUSCULAS Y MINUSCULAS

En general, BASIC puede trabajar con letras mayúsculas o minúsculas. BASIC convierte, automáticamente las palabras reservadas y variables a letras mayúsculas. Las únicas veces que se conservan los caracteres minúsculas escritos por el usuario son: 1. Caracteres encerrados entre comillas; 2. Caracteres formando parte de un comentario (esto es, en líneas con sentencias comentario REM; se estudiarán en el capítulo 3).

En el libro utilizaremos para mensajes entre comillas o dentro de sentencias REM, indistintamente mayúsculas o minúsculas. Las palabras reservadas se escribirán siempre en mayúsculas como lo hace BASIC aunque el usuario puede teclearlas en minúsculas, pues la máquina normalmente lo entenderá, aunque luego al listarlas con la orden LIST se presentarán en mayúsculas.

Como ya habrá averiguado la escritura de mayúsculas se consigue pulsando cualquier tecla SHIFT (⇧) simultáneamente con la tecla del carácter deseado, o fijando la escritura continua de caracteres mayúsculas con la tecla CAPS LOCK o CAPS SHIFT.

Para afianzar estos conceptos, escriba el siguiente programa:

```
10 print "Hola forastero, yo de ti no lo haria"
```

pulse la tecla RETURN

```
20 beep
```
(BEEP sentencia que emite un pitido por altavoz)

pulse de nuevo RETURN. El programa completo quedaría en pantalla así:

```
10 print "Hola forastero, yo de ti no lo haria"
20 beep
```

Borre la pantalla introduciendo CLS y naturalmente pulsando RETURN. El programa, sin embargo, permanece en la memoria hasta que apague la computadora naturalmente.

Ahora introduzca

```
list
```

se visualizará a continuación:

```
10 PRINT "Hola forastero, yo de ti no lo haria"
20 BEEP
OK
```

Las palabras PRINT y BEEP escritas en minúsculas se han representado en mayúsculas. La orden LIST siempre transforma en mayúsculas las palabras reservadas, con independencia de cómo se hayan introducido. La ventaja de esta característica es que su programa si lista de modo claro y transparente diferenciando las palabras reservadas, mientras que el usuario se puede ahorrar la escritura en mayúsculas y el inconveniente que supone la pulsación de las teclas SHIFT o CAPS SHIFT.

Como ya se ha comentado en este libro, *se escribirán todas las palabras reservadas en mayúsculas, aunque usted puede introducirlas por teclado en minúsculas, normalmente.*

EJECUCION DE UN PROGRAMA

La ejecución o lanzamiento de un programa se realiza mediante la orden RUN. Tan pronto como se escriba RUN y se pulse la tecla RETURN se ejecuta el programa. A esta acción también se la conoce por *«correr el programa».*

El *formato* de la orden RUN es:

RUN *espacio en blanco o número de línea*

Si se pulsa RUN, comienza la ejecución en el número de línea más bajo, y si se hace RUN n, comienza la ejecución en el número de línea n.

Sea el programa DEMO 1 ya visto y que tiene introducido en memoria. Realize a continuación:

```
RUN
**********FIN
```

Resultado en pantalla de la ejecución DEMO 1 con RUN

```
RUN 40
FIN
```

Resultado en pantalla de la ejecución DEMO 1 con RUN 40

MODIFICACION DE UN PROGRAMA

Una vez introducido un programa en memoria, es preciso ejecutarlo y comprobar si ha funcionado correctamente o si se desea mejorar. A esta fase se le conoce como *depuración del programa* y permite la corrección y puesta a punto del mismo, hasta dejarlo preparado para su correcto funcionamiento.

La fase de depuración comprende la *ejecución, modificación de líneas, listado de programa, inserción* y *añadido de líneas, borrado de líneas* y *nueva ejecución* hasta su funcionamiento correcto.

El programa *editor* (más tarde se volverá a hablar de él) permite que la modificación de un programa se realice del modo más eficaz posible. La potencia del editor es una de las características más notables de una computadora, dado que hasta la ejecución correcta de un programa es preciso pasar por la fase de introducción y depuración, que normalmente conlleva un gran número de horas de trabajo y manejo del teclado de la computadora.

Así, un buen editor de pantalla permitirá que el cursor se pueda desplazar por toda su superficie en las cuatro direcciones.

MODIFICACION DE UNA LINEA DE PROGRAMA EXISTENTE

Supongamos que en el programa DEMO 1 se desea modificar la línea 10 por una nueva línea 10 FOR I=1 TO 30. Bastará escribir 1- FOR I=1 TO 30 y pulsar RETURN.

Otro sistema sería listar la línea deseada mediante la orden LIST 10 o EDIT 10 y, por supuesto, pulsar RETURN, con lo que aparecerá en pantalla:

```
10 FOR I=1 TO 10
```

y a continuación, mediante el adecuado movimiento de cursor o teclas de edición se modificará la línea, cambiando el último número 10 por 30. Después se pulsará RETURN.

De este modo queda modificada dicha línea.

LISTADO DE UN PROGRAMA

En numerosas ocasiones es preciso *listar* el programa que está actualmente en la memoria de la computadora, parcial o completamente bien en la pantalla del terminal o monitor, o bien en la impresora.

El examen del listado del programa permitirá analizar el mismo para la modificación, corrección o conservación del mismo. El formato de la sentencia o comando es:

LIST	*espacio en blanco*	listado del programa completo
	número de línea n	listado de la línea n
	n.º $línea_1$ *—n.º* $línea_2$	listado desde la $línea_1$ a la $línea_2$
	—n.º línea	listado desde el principio a una línea en concreto
	n.º línea—	listado desde una línea en concreto hasta el final

Así en el programa DEMO 1 se pueden dar los casos siguientes:

Escriba LIST <RETURN>

se visualiza

```
5 REM DEMO 1
10 FOR I=1 TO 10
20 PRINT "*";
30 NEXT I
40 PRINT "FIN"
50 END
```

LIST 20 <RETURN>

```
20 PRINT "*";
```

LIST 5-30 <RETURN>

```
5 REM DEMO 1
10 FOR I=1 TO 10
20 PRINT "*";
30 NEXT I
```

LIST -30 <RETURN>

```
5 REM DEMO 1
10 FOR I=1 TO 10
20 PRINT "*";
30 NEXT I
```

LIST 30- <RETURN>

```
30 NEXT I
40 PRINT "FIN"
50 END
```

Si el programa es muy grande y no cabe por completo en la pantalla, el listado aparecerá a gran velocidad en la misma, lo que impedirá su visualización total o parcial con detenimiento. Existe en ocasiones una tecla que permite ralentizar la presentación o desplazar el listado a voluntad.

Algunas computadoras poseen una tecla denominada ROLL o SCROLL que permite desplazar el listado en la pantalla hacia arriba o hacia abajo, en una acción que se denomina *enrrollamiento* (*scrolling* en inglés).

INTERRUPCION DEL LISTADO

Se puede interrumpir un listado antes de llegar al final pulsando la tecla STOP, BREAK o PAUSE. Existen, sin embargo, diferencias entre unos modelos de computadoras y otros:

IBM PC:

— Pulsación de las teclas CTRL y NUM LOCK

- Pausa temporal, se reanuda el listado pulsando cualquier otra tecla.

— Pulsación de las teclas CTRL y BREAK

- Aborta el listado, detiene el listado definitivamente.

Amstrad:

— Pulsación de la tecla ESC una vez
- Pausa temporal, se reanuda el listado pulsando cualquier tecla.

— Pulsación de la tecla ESC dos veces
- Se detiene el listado definitivamente.

MSX:

— Pulsación teclas CTRL y STOP.

EDICION DE LINEAS

Algunas versiones de BASIC (MBASIC, BASICA, Amstrad, HEWLETT PACKARD, etc.) permiten un procedimiento de modificación de líneas, EDIT LINE (LINEA A EDITAR) muy similar a LIST y que facilita considerablemente la edición de programas. Se introduce el modo edición (EDIT) con la orden EDIT.

La orden EDIT se puede utilizar también para hacer cambios en una línea específica de un programa. Ahorra el movimiento del cursor a la línea en cuestión con las teclas de movimiento de cursor que son especialmente útiles, para largos listados.

El formato es:

EDIT *línea*

Por ejemplo, en la línea 20 del programa DEMO 1 deseamos cambiar el signo punto y coma (;) por una coma (,).

Escriba y pulse, como siempre, a su terminación ENTER

```
EDIT 20
```

BASIC imprime en la pantalla la línea solicitada y coloca el cursor bajo el primer dígito del número de línea

```
20 PRINT "*";
-
```

corrija ahora la línea con las teclas de edición correspondientes.

La orden EDIT es como LIST, sirve para visualizar su programa en la pantalla a fin de poder hacer modificaciones. Sin embargo, LIST permite listar o visualizar más de una línea de programa a la vez.

Si utiliza

```
EDIT .
```

se visualiza la línea que se acaba de introducir en memoria.

BORRADO DE LINEAS DE PROGRAMA

En la mayoría de las versiones BASIC, el borrado o eliminación de líneas de memoria se hará simplemente tecleando el número de línea e inmediatamente RETURN.

En el programa DEMO se borran las líneas 5, 10 y 40 del modo siguiente:

```
5   <RETURN>
10  <RETURN>
40  <RETURN>
```

Para comprobar su eliminación de memoria, liste el programa escribiendo LIST y pulse RETURN.

```
20 PRINT "*";
30 NEXT I
50 END
```

Sin embargo, si el número de líneas a borrar es elevado e incluso muchas de ellas son consecutivas, el proceso resulta laborioso a la par que tedioso. Por ello los BASIC avanzados (hoy en día la mayoría) poseen la orden DELETE o DEL (abreviatura de DELETE, «eliminar, borrar») que funciona de modo similar a la orden LIST.

DEL N	*Borra línea n.*
DEL N1-N2	*Borra las líneas comprendidas entre n_1 y n_2.*
DEL -N1	*Borra todas las líneas desde el principio a la línea n_1 inclusive.*
DEL N1-	*Borra todas las líneas desde n_1 al final.*

Así, en el programa DEMO original, si desea borrar las líneas 5, 10 y 40 se procedería del modo siguiente:

```
DEL 5-10   <RETURN>
DEL 40     <RETURN>
```

NOTA: En computadora IBM PC y compatibles existen otros métodos para eliminación de líneas de la memoria:

1. Cancelación de una línea

Existe otro medio para eliminar una línea no deseada; consiste simplemente en situar el cursor en cualquier parte de la línea en cuestión y pulsar la tecla ESC («escape») que está localizada cerca de la esquina superior izquierda del teclado. Si se pulsa ENTER después de pulsar ESC borrará la línea completa de memoria.

2. Borrado parcial de una línea

Se puede truncar o eliminar la parte derecha (de cualquier longitud) de una línea; para ello lleve el cursor con las teclas de movimiento de cursor al primer carácter que desea borrar y a continuación pulse las teclas CTRL y END, con lo que se borrará este carácter y todos los situados a su derecha.

ADICION O INSERCION DE LINEAS DE PROGRAMA

Para introducir nuevas líneas de programa bastará teclear el número de línea y la/s sentencia/s correspondientes y a continuación RETURN. La computadora mezclará las líneas, de modo automático, con las que ya tiene en memoria y en el orden adecuado.

INSERCION DE LINEAS DE PROGRAMA

En algunas ocasiones cuando la adición de líneas supone una inserción entre dos líneas ya existentes suele haber en algunas computadoras (Sistema 45 de Hewlett Packard) una tecla denominada INSERT LINE que trabaja insertando líneas en el programa.

Listando o editando el programa, basta buscar la línea inmediatamente superior a la que se desea insertar y colocar en ella el cursor, pulsando a continuación la tecla INSERT LINE, lo que permite introducir la sentencia deseada. (Dado que no es muy usual esta tecla, no insistiremos sobre ella.)

NUMERACION Y RENUMERACION AUTOMATICA DE LINEAS: AUTO Y RENUMBER

Los programas editores modernos de BASIC poseen órdenes para la numeración y renumeración automática de líneas de programa. Las órdenes AUTO y RENUMBER facilitan la edición de programas:

Orden **AUTO:** Al introducir esta orden (o pulsar la tecla correspondiente si existe), los números de línea se van numerando de modo automático a medida que se ingresan y almacenan en la memoria, con lo que se evita numerarlas por teclado, con la siguiente disminución de los errores que su escritura puede producir.

Formato: AUTO *número de línea inicial, valor del incremento.*

Si no se especifica el número de línea inicial ni el incremento, la numeración comienza de modo automático en la línea 10, y una vez introducida la sentencia completa y pulsando RETURN aparecerá en pantalla 20, luego 30, 40, y así sucesivamente.

AUTO	*Numera las líneas de modo automático comenzando en la línea 10 (algunas versiones podrían ser 100) y con incrementos de valor 10.*
AUTO N	*Numera las líneas de modo automático comenzando en la línea n y con incrementos de 10 en 10.*
AUTO N1, N2	*Numera las líneas de modo automático comenzando en la línea n y con incrementos de n_2 en n_2.*

Si ha comenzado la escritura de su programa sin el generador automático de líneas (orden AUTO) y desea iniciar una numeración automática de su línea actual, la orden

```
AUTO.,10
```

arrancará desde línea actual incrementándose de 10 en 10, ya que el punto (.) se utiliza como símbolo para el número de línea actual.

Si el generador de líneas se utiliza para generar un número de líneas que ya existe en la memoria de la computadora, un asterisco (*) se imprimirá después del número de línea, para indicar que, cuando introduzca una nueva línea, esta nueva línea sustituirá a la antigüa línea de igual número. Si pulsa la tecla RETURN después de la posición del carácter * la línea existente no se modificará.

Para salir del modo AUTO, se han de pulsar las teclass CTRL y BREAK o alguna de las ya vistas para interrupción de listados como BREAK, ESC, etc. Aparecerá el conocido mensaje *OK,* significándole al usuario que el PC se encuentra en modo directo. Nótese que cuando se pulse CTRL-BREAK, la línea en que se está trabajando no se almacena como parte del programa actual.

Orden **RENUMBER, RENUM, REN**

La orden RENUM, o su abreviatura REN, o bien la palabra inglesa completa RENUMBER, permite renumerar las líneas de un programa queya está en memoria. Esta orden es de gran utilidad sobre todo en la fase de depuración del programa o a la terminación del mismo, al objeto de poder conservar el programa y trabajar con una numeración más adecuada y flexible.

El *formato* de esta orden es:

RENUM *número de la línea inicial, primera línea valor del incremento.*

RENUM, RENUMBER, REN (abreviaturas más usuales de *renumber*)

RENUM	*Renumera todo el programa, de modo que a la primera línea existente se le asigna el valor 10 (algunas versiones podrían ser 100) y las líneas siguientes se numeran en incrementos de 10 en 10.*

`RENUM N1`	*Renumera el programa completo comenzando en N_1 y aumentando de 10 en 10.*
`RENUM N1, N2`	*Renumera un programa comenzando con N_1 y aumentando de N_2 en N_2.*
`RENUM N1, N2, N3`	*Renumera un programa comenzando en la línea N_3 y numerándose a partir de N_1 y de N_3.*
`RENUM N1, N2, N3, N4`	*Renumera las líneas antiguas de un programa desde N_3 a N_4. La línea N_3, se asigna a N_1 y las siguientes se incrementan en N_2, terminando en N_4.* (Sólo la utilizan escasas computadoras.)

La orden RENUM renumera todas las líneas del programa incluyendo los números de líneas de las instrucciones de control GOTO, GOSUB, THEN, ELSE, ON-GOTO, ON-GOSUB, RESTORE, RESUME y ERL.

Ejercicio práctico de edición

Supongamos que deseamos introducir en la computadora el siguiente programa:

```
10 REM DEMO 2
20 A=5
30 B=4
40 C=6:D=8
50 M=(A+B+C+D)/X
60 PRINT M
70 GOTO 200
80 PRINT
90 PRINT
100 PRINT "FIN"
110 END
```

El procedimiento más adecuado sería el siguiente:

1. *Teclear* `AUTO` <RETURN>

Se visualizará	`10`	*se introduce*	`REM DEMO 2`	<RETURN>
Se visualizará	`20`	*se introduce*	`A=5`	<RETURN>
Se visualizará	.	*se introduce*	.	.
Se visualizará	.	*se introduce*	.	.
Se visualizará	.	*se introduce*	.	.
Se visualizará	.	*se introduce*	.	.
Se visualizará	`110`	*se introduce*	`END`	<RETURN>
Se visualizará	`120`	*se pulsa* BREAK *o similar*		Aparece mensaje OK o Ready

2. Tras su introducción nos damos cuenta que se nos olvidó incluir una línea entre 10 y 20 que represente PRINT "INTRODUZCA VALOR DE X", y otra que sea INPUT X. O sea, por ejemplo:

```
12 PRINT "INTRODUZCA VALOR DE X"
15 INPUT X
```

3. Se escriben e introducen las líneas 12 y 15 del modo conocido.
4. Se lista el programa completo:

`LIST` <RETURN>

```
10 REM DEMO 2
12 PRINT "INTRODUZCA VALOR DE X"
15 INPUT X
20 A=5
30 B=4
40 C=6:D=8
50 M=(A+B+C+D)/X
60 PRINT M
70 GOTO 200
80 PRINT
90 PRINT
100 PRINT "FIN"
110 END
```

5. Se ejecuta el programa mediante RUN; aparecerá en la pantalla un signo de interrogación que significa que ha de introducir un valor numérico que será el divisor del cociente A+B+C+D, y a continuación pulse RETURN; se imprimirá inmediatamente el resultado del cociente buscado.

RUN

? *(Escriba por ejemplo 10 y pulse* <RETURN>*) 10*
 2.3

6. Tras comprobar que funciona correctamente, siga adelante.

7. Ahora desea renumerar el programa, de modo que la primera línea sea 100 y que sus incrementos de línea sean 10.
   Escriba RENUM 100, 10 y pulse RETURN.

8. Compruebe la renumeración.

   Pulse LIST seguido de RETURN.
   Se visualizará el siguiente programa:

```
100 REM DEMO 2
110 PRINT "INTRODUZCA VALOR DE X"
120 INPUT X
130 A=5
140 B=4
150 C=6:D=8
160 M=(A+B+C+D)/X
170 PRINT M
180 GOTO 220
190 PRINT
200 PRINT
210 PRINT "FIN"
220 END
```

NOTA: *Este programa no tiene más objeto que ejercitar al alumno en el manejo del teclado y de las órdenes de edición AUTO, RENUM. En consecuencia los resultados de su ejecución no tienen importancia.*

INTERRUPCION DE UN PROGRAMA (BREAK/STOP)

Se puede detener un programa mientras se está ejecutando. A este acción se la conoce como interrupción del programa (BREAK, BREAK/STOP en inglés y en terminología de programación).

La interrupción del programa se realiza mediante la orden STOP (BREAK en algunas computadoras) De modo similar al explicado en el parrafo INTERRUPCION DEL LISTADO.

Así pues, la misma tecla o secuencias de teclas que detenía o interrumpía el listado hace lo mismo con el programa.

Ejemplos:

Introduzca y ejecute el siguiente programa:

```
100 FOR I=1 TO 30
110 PRINT I;
120 NEXT I
RUN
```

Pulse CTRL-BREAK (ESC, CTRL STOP, etc.) en un momento dado de la impresión de resultados

```
1  2  3  4  5  6  7  8  9  10  11  12  13  14
Break in 110
Ok
CONT
15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
Ok
```

Introduzca y ejecute el siguiente programa:

```
5 REM DEMO 1
10 FOR I=1 TO 30
20 PRINT "*";
30 NEXT I
40 STOP
50 PRINT "FIN"
60 END

RUN
******************************
Break in 40
Ok
CONT
FIN
Ok
.
```

Pulsación de CTRL-BREAK

La *interrupción* de un programa se puede hacer por dos procedimientos:

a) desde teclado; *b)* desde dentro del propio programa.

Mediante teclado, como se ha visto, se consigue por la pulsación simultánea de las teclas CTRL y STOP, CTRL y STOP, ESC, etc.

Desde el interior de un programa se obtiene con la sentencia STOP, cuya finalidad es detener el programa y poner a la computadora en modo directo.

Formato:



La sentencia STOP se puede utilizar en cualquier parte de un programa para terminar su ejecución. Cuando BASIC encuentra una sentencia STOP visualiza el mensaje:

```
BREAK IN n
```

donde *n* es el número de línea donde se sitúa STOP.

Se puede reanudar la ejecución del programa, realizando la orden CONT (escritura de CONT y pulsación de la tecla RETURN). La ejecución continúa en el punto donde ocurrió la interrupción. CONT no es válido si el programa se ha editado durante la interrupción. Sin embargo, siempre actúa, tanto con la sentencia STOP como con la pulsación de las teclas de interrupción.

PAUSA DURANTE LA EJECUCION

Existe otra orden/sentencia/secuencia de pulsación de teclas que detiene temporalmente la ejecución de un programa: es la función PAUSA que en algunas computadoras se realiza con una tecla denominada PAUSE.

Cuando se pulsa la tecla PAUSE, se termina de ejecutar la instrucción en curso y el programa se detiene al llegar a la línea siguiente. De igual modo se puede programar usando la sentencia PAUSE.

La detención de un programa interrumpido mediante STOP o PAUSE se continúa pulsando la orden CONT. El programa continúa en el punto exacto en que ocurrió la interrupción.

La función *pausa* se invoca en las computadoras IBM PC presionando las teclas CTRL y NUM LOCK al mismo tiempo y en las computadoras Amstrad pulsando la tecla ESC. La función «pausa» detiene la ejecución del programa temporalmente. El programa se reanudará pulsando cualquier otra tecla.

Esta función es útil cuando se están visualizando datos a gran velocidad y el operador no puede leerlos detenidamente. De este modo el programa se puede detener o interrumpir a voluntad del usuario, de modo que se puedan estudiar pausadamente datos, resultados, listados, etc.

4. EL EDITOR DE PANTALLA

El editor, como ya se ha visto, es un programa que permite escribir líneas de texto y en particular líneas de sentencias BASIC correspondientes a un programa.

La potencia de un editor es una de las características notables a tener en cuenta a la hora de elegir la computadora de trabajo. Tenga presente que el mayor tiempo en desarrollo de programas BASIC se emplea ante la pantalla y escribiendo o modificando sus programas.

Un buen editor, además de permitir la introducción de líneas, borrado, numeración y renumeración automática de las mismas, ha de permitir modificar total o parcialmente y a voluntad la/s línea/s de programa que así se decidan.

En la mayoría de los sistemas de microcomputadoras las funciones de edición están disponibles directamente en el teclado y son fundamentalmente las siguientes:

a) *Movimiento de cursor.*
b) *Inserción y borrado de caracteres.*
c) *Ordenes asociadas a la tecla* CTRL (CONTROL).
d) *Ordenes especiales.*

A estas citadas habría que añadir las órdenes AUTO, RENUMBER, DELETE y EDIT ya comentadas.

No es posible recoger aquí las diversas características de los diferentes editores existentes, dado que ésta es una característica diferenciadora propia de las computadoras. En consecuencia, hemos optado por examinar un teclado estándar lo más potente posible y a continuación ir desdoblando sucesivas teclas indicándoles cuáles de ellas existen en las computadoras que se consideren en el curso.

De esta forma creemos que el lector no sólo podrá familiarizarse con el teclado de su computadora, sino con otros disponibles en el mercado.

4.1. El teclado

Los teclados de computadoras o terminales se utilizan para introducir sentencias, programas y datos requeridos por los mismos. El tipo de teclado depende del modelo de computadora, pero en general todos suelen ser del tipo QWERTY, semejante al de una máquina de escribir.

En general, las teclas se pueden agrupar en los siguientes grupos:

— *Teclas alfabéticas*

Todas las computadoras suelen disponer de dos juegos de letras: mayúsculas y minúsculas, pasándose de unas a otras mediante la pulsación de la tecla SHIFT o de otra similar. Suelen contener las 26 letras, de la A a la Z (el Gobierno español dictó el año 1985 un decreto de homologación de teclados que exige a todos los fabricantes vender los teclados con el alfabeto español, ñ, acentos, etc.).



Figura 1.1. Teclado de las computadoras IBM PC y XT.

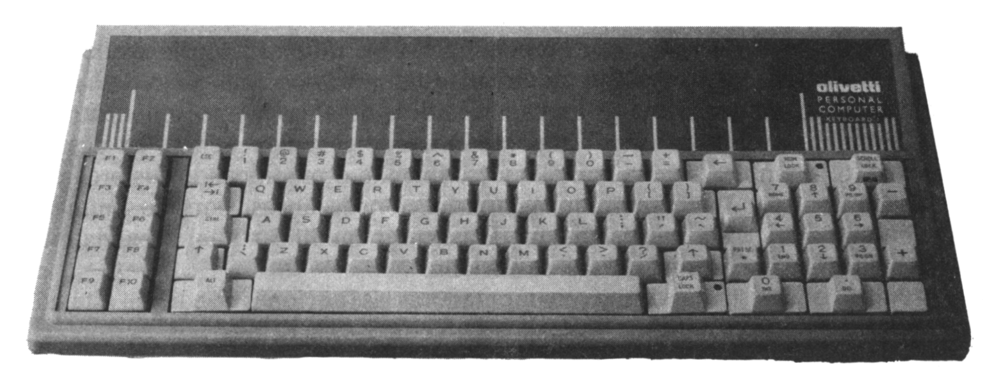


Figura 1.2. Teclado de la computadora Olivetti M24.

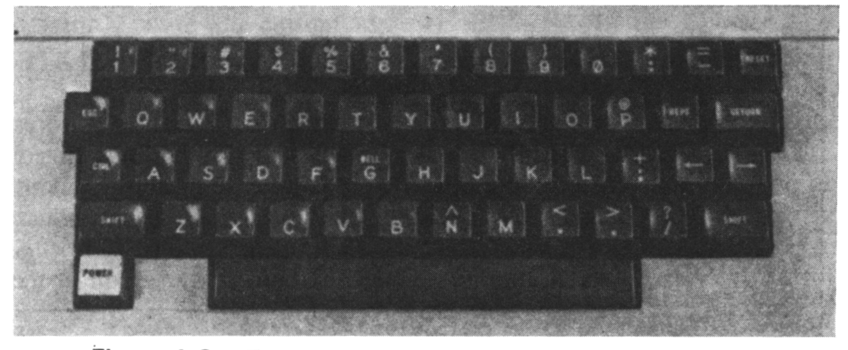


Figura 1.3. Teclado de la computadora Apple IIe.

— *Teclas numéricas*

Proporcionan los dígitos 0 a 9. Aquellos teclados que poseen una parte exclusiva como teclado numérico, además de los números suelen incluir los operadores aritméticos: +, −, *, /, > o < y símbolos tales como (.), (,), (').

— *Teclas de signos especiales*

Incluyen los símbolos y caracteres especiales usados con frecuencia, #, $, %, *, etc., así como los dígitos 0 a 9.

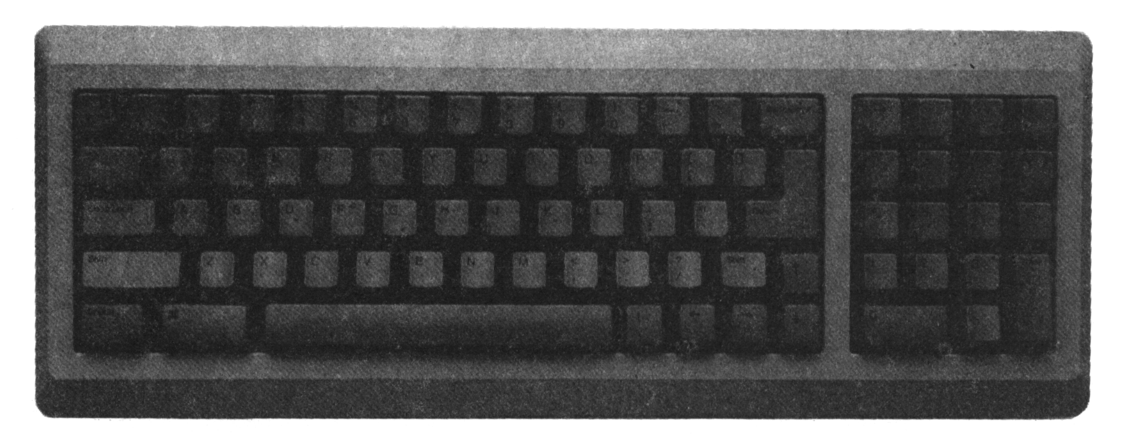


Figura 1.4. Teclado de la computadora Macintosh.

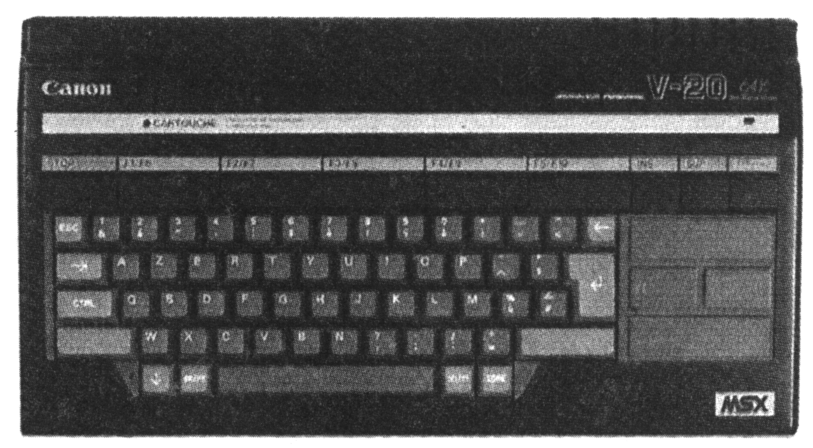


Figura 1.5. Teclado de una computadora MSX.



Figura 1.6. Teclado de la computadora Amstrad CPC 464.

— *Teclas de función*

SHIFT (↑). Tecla de cambio. Se emplean con las letras para pasar de mayúsculas a minúsculas o bien para representar las órdenes o caracteres que están en la parte superior de las teclas (cumple igual función que su equivalente en una máquina de escribir).

SHIFT LOCK. Fijación de mayúsculas/minúsculas o caracteres, órdenes de la parte superior de la tecla.

CTRL. Se utiliza en unión de otras teclas para definir caracteres, colores o características especiales de la computadora.

RETURN, ENTER, NEW LINE, CR, STORE, (↵). Es la tecla equivalente al retorno de carro en una máquina de escribir y mediante su pulsación se introducen las órdenes e instrucciones a la memoria para que puedan ser interpretadas y ejecutadas por la computadora.

ESC. Tiene diferentes funciones según el modelo de compuadora. En Amstrad interrumpe temporalmente el programa de ejecución; en IBM PC borra la línea donde se encuentra el cursor.

ALT. Similar a CTRL.

— *Teclas de función especiales o programables*

Algunas computadoras (HP-85/86/87, IBM PC, MSX, Amstrad, Commodore 64, etc.) poseen unas teclas de función que pueden ser programadas por el usuario o son programadas ya de antemano por el intérprete de BASIC. De esta forma, si por ejemplo una computadora posee cuatro teclas de función programables, se podría hacer el siguiente juego:

```
F0      LIST
F1      PRINT
F2      INPUT
F3      RUN
```

Bastaría pulsar F_0, F_1, F_2, F_3, para que en pantalla apareciesen las palabras o sentencias LIST, PRINT, INPUT, RUN. De este modo se ahorraría tiempo en la escritura de los programas.

— *Modelos de teclas de función en diversas computadoras*

IBM PC

```
F1      LIST
F2      RUN    <RETURN>
F3      LOAD"
F4      SAVE"
F5      CONT
F6      ,"LPT 1:"    <RETURN>
F7      TRON    <RETURN>
F8      TROFF    <RETURN>
F9      KEY
F10     SCREEN 0,0,0    <RETURN>
```

MSX

```
F1      COLOR
F2      AUTO
F3      GOTO
F4      LIST
F5      RETURN
```

En las computadoras que poseen líneas teclas de función, la sentencia KEY LIST, visualiza en pantalla todas las funciones obtenidas al pulsar las correspondientes teclas. Las sentencias KEY ON y KEY OFF borran o presentan los títulos de las funciones, si ellos se pueden visualizar en la línea inferior de la pantalla.



Figura 1.7. Teclado de la computadora ZX Spectrum Plus.

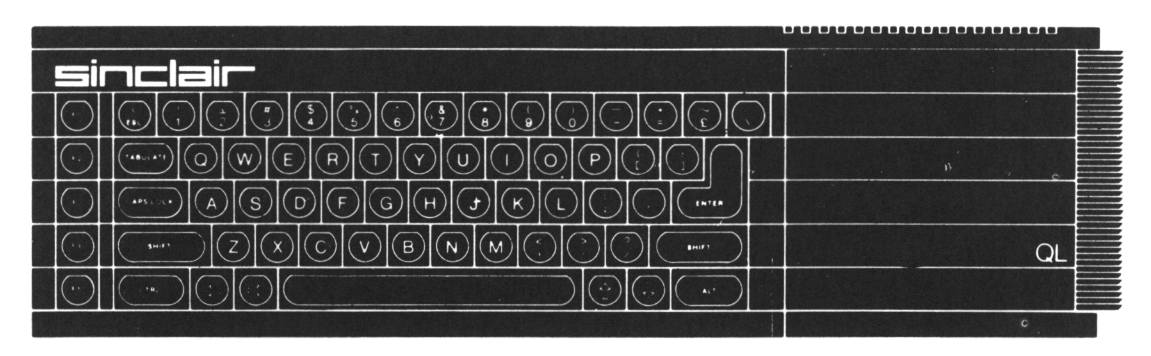


Figura 1.8. Teclado de la computadora Sinclair QL.

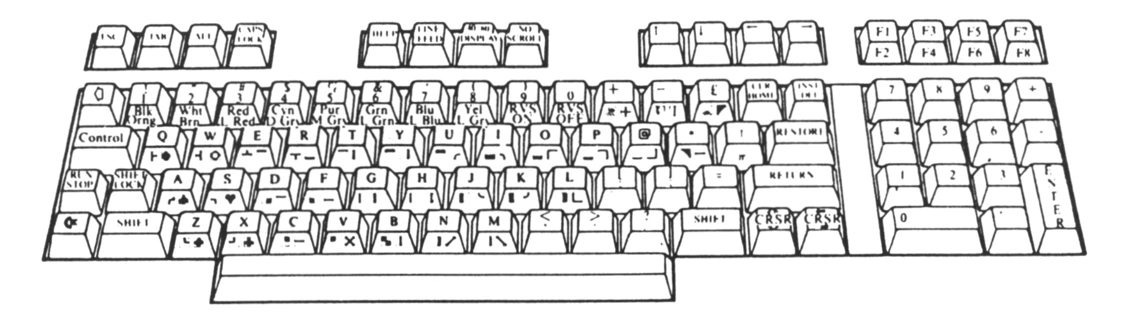


Figura 1.9. Teclado de la computadora Commodore 128.

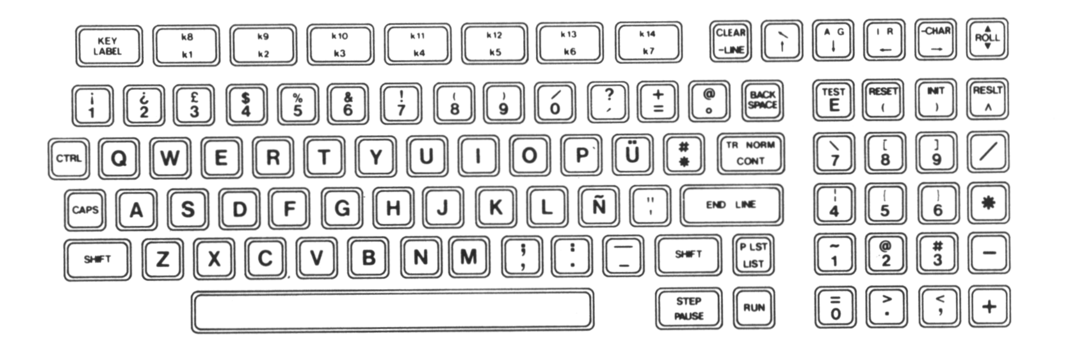


Figura 1.10. Teclado español de la computadora HP-86B.

— *Teclas de control de edición de pantalla*

Son las teclas propiamente de edición: movimiento de cursor, borrado de pantalla, etc.

INSERT	*Permite insertar caracteres.*
DELETE/BACKSPACE (←)	*Permite borrar caracteres.*
HOME	*Lleva el cursor a la posición inicial (parte superior izquierda de la pantalla).*
CLEAR o CLS	*Borrado de la pantalla.*
ROLL (SCROLLING)	*Recupera información que se ha desplazado fuera de la pantalla. Al pulsar ROLL, la información que se acaba de salir de la pantalla desciende, mientras que al pulsar ROLL y SHIFT, asciende la información últimamente almacenada en la memoria pantalla.*
REVERSE/ON/OFF	*Activa el campo inverso de presentación permitiendo la escritura sobre dicho fondo.*
BLINKING	*Activa y permite el parpadeo de letras o caracteres.*
UNDERLINE	*Permite el subrayado de mensajes.*
STOP	*Interrupción del programa.*

— *Teclado especial*

Sólo se incluye en computadoras personales potentes y suele constar de una columna simple, doble o triple de teclas que incluye las órdenes más usuales que se dan a la máquina, tales como RUN, CONT, PAUSE, STOP, LOAD, etc. Facilita considerablemente la edidión y ejecución de programas.

— *Teclas de control de cursor*

Normalmente el cursor se puede mover por toda la pantalla en las cuatro direcciones y es una característica muy notable. Actualmente todas las computadoras poseen este juego de teclas.

En COMMODORE existen dos teclas $\overset{\longrightarrow}{\underset{\longleftarrow}{\text{CRSR}}}$ y $\overset{\uparrow}{\underset{\downarrow}{\text{CRSR}}}$ que si se pulsan en modo normal se desplaza el cursor a la derecha y abajo un espacio cada vez que se pulsan, y si se oprime en modo desplazado (mediante la tecla SHIFT) el cursor se desplaza a la izquierda y arriba.

En APPLE II se han de pulsar dos teclas en secuencia. Primero pulse la tecla ESC (escape) y después la tecla A, B, C o D, con lo que se desplazará el cursor una posición a derecha, izquierda, abajo o arriba.

(En otras computadoras suelen existir algunas de estas teclas u otras especiales que tienen carácter repetitivo y que permiten retrocesos o avances rápidos.)

En APPLE II, además de las funciones vistas para la tecla ESCAPE, suele tener asociadas algunas otras:

ESC_E Borra una línea desde la posición de cursor hasta el final de la línea.

ESC_F Borra la pantalla desde la posición del cursor hasta el fondo de la pantalla.

$ESC_@$ Borra toda la pantalla y lleva el cursor a posición inicial.

En IBM PC y compatibles, MSX y Amstrad existen 4 teclas para movimientos de cursor.

— *Ordenes asociadas a la tecla* CONTROL (CTRL)

La tecla de CONTROL (o CTRL), que puede venir incorporada en todas las computadoras modernas, no tiene estandarizada su función, si bien suele utilizarse siempre en unión de otras teclas.

Así, en el Commodore 64, al pulsar CTRL y las teclas de color de pantalla, REVERSE ON, REVERSE OFF, se activa el color correspondiente de la pantalla, se activa o desactiva el vídeo inverso.



Figura 1.11. Teclado de la computadora Olivetti M 24-3270.

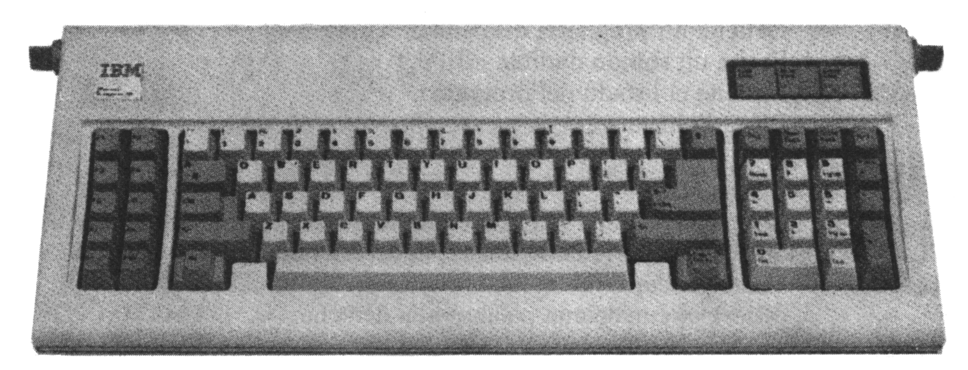


Figura 1.12. Teclado de la computadora IBM AT.

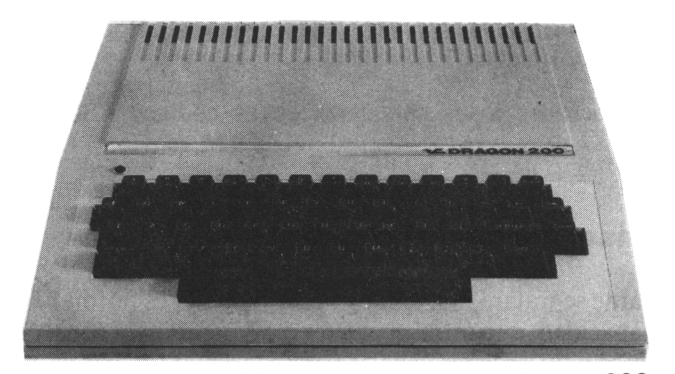


Figura 1.13. Teclado de la computadora Dragon-200.



Figura 1.14. Teclado de la computadora Commodore 128.



Figura 1.15. Teclado de la computadora Secoinsa FM7.

En el APPLE II:

CTRL-B	Sale del programa monitor hacia el BASIC perdiéndose el programa.
CTRL-C	Detiene un programa que está en curso.
CTRL-G	Emite un sonido audible «BIP».
CTRL-S	Detiene el listado del programa.
CTRL-X	Borra la línea sobre la que se encuentra el cursor.

En el IBM PC:

CTRL-HOME	Limpia la pantalla y lleva el cursor a la posición inicial o casa («home»).
CTRL-END	Se borra la parte de la línea a la derecha del cursor.
CTRL-←	Mueve el cursor una palabra a la izquierda.
CTRL-→	Mueve el cursor una palabra a la derecha.

En MSX:

CTRL-STOP	Se interrumpe la ejecución del programa.
CTRL-tecla	Realiza diferentes funciones (tecla cualquier tecla del teclado).

En Amstrad:

Esta tecla permite acceder a los caracteres cuyos códigos ASCII están comprendidos entre 1 y 31.

4.2. Carga y conservación de programas

Aunque más adelante se tratará este tema en profundidad, es normal, sobre todo en computadoras de pequeña capacidad, que utilicen casetes o disquetes para conservar o guardar programas. Por ello, incluimos aquí órdenes usuales para *conservar* (grabar en memoria tipo disco o casete) o *cargar* (transferir el programa del casete o disco a la memoria central).

LOAD *"nombre del programa"*

— Carga el programa almacenado en el casete o disco a la memoria de la computadora.

SAVE *"nombre del programa"*

— Conserva en la memoria de casete o disco el programa almacenado en la memoria de la computadora.

Estos formatos, al igual que cualquier orden o sentencia, deben ir acompañados de la pulsación de RETURN.

LOAD *"nombre del programa"* <RETURN>
SAVE *"nombre del programa"* <RETURN>

En el caso de unidades de casetes, la sentencia LOAD" " carga el primer programa que se encuentra en la cinta de casete.

Las órdenes LOAD y SAVE, pueden tener diversos formatos según las computadoras.

Amstrad

LOAD "DEMO"	*Carga el programa DEMO desde la unidad de casete.*
LOAD ""	*Carga el primer programa de la cinta.*
SAVE "PROG"	*Graba o guarda el programa bajo el nombre PROG.*
SAVE ""	*Graba sin dar nombre.*
SAVE "PROG",P	*Graba bajo forma protegida, impidiendo la visualización del listado.*
SAVE "PROG",A	*Graba en formato ASCII o de texto.*

MSX

LOAD "PROG"	*Carga el programa "PROG".*
LOAD "PROG",R	*Carga el programa y lo ejecuta a continuación.*
SAVE "DEMO"	*Graba el programa DEMO.*
SAVE "DEMO",A	*Graba el programa DEMO en formato ASCII.*

IBM PC

LOAD "B:DEMO.BAS"	*Carga el programa DEMO.BAS contenido en la unidad de disco B.*
LOAD "B:DEMO.BAS",R	*Carga el programa DEMO.BAS y lo ejecuta inmediatamente.*
SAVE "DEMO",A	*Graba el programa DEMO en la unidad de disco con el formato ASCII.*
SAVE "DEMO",P	*Graba el programa en formato protegido, que impedirá su listado, en la unidad de disco.*

Commodore, Apple y las versiones MBASIC

Los formatos son similares a los anteriores.

4.3. Carga y ejecución automática de programas

La mayoría de las computadoras que soportan unidades de disco, poseen órdenes de carga y ejecución automática de programas de las unidades de disco. Su formato es

RUN *"nombre de programa"*

RUN "DEMO" *Se carga en memoria el programa DEMO almacenado en la unidad de disco y a continuación se ejecuta.*

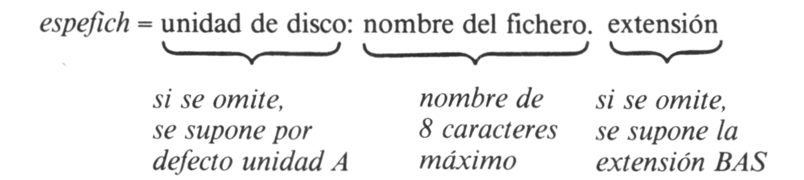
5. PROCESAMIENTO DE FICHEROS (MBASIC, BASICA/IBM PC)

Un *fichero* es un programa o una colección de datos como se verá en el capítulo 11. Los programas se conservan en disco con la orden SAVE y a partir de ese instante se denominan ficheros. Los ficheros pueden ser de *programas* o *datos* según almacenen programas o datos; estos últimos sólo pueden ser tratados por programas.

5.1. Nombres de ficheros

MBASIC posee órdenes que le serán de gran utilidad para administrar ficheros. Las órdenes FILES, KILL y NAME. Se refieren a la obtención del directorio —listado y fichero almacenados en disco—, borrado de ficheros del disco y cambio de nombre de un fichero grabado en disco. Las órdenes/sentencias MERGE y CHAIN son útiles cuando se desea mezclar o conectar dos o más programas BASIC. Las órdenes SAVE, LOAD y RUN administran los programas del modo ya conocido.

Veamos el concepto de *espefich* (especificador de fichero) antes de pasar a explicar las diferentes órdenes.



Ejemplos de *espefich* son:

```
A:DEMO1.BAS          B:NOMINAS.BAS
TEST                 B:TEST
TEST.MAC             B:TEST.PIP
```

5.2. FILES. Visualización del directorio

La orden FILES visualiza todos los ficheros del disquete situados en la unidad de disco por defecto. Actúa como la orden DIR o CAT de los sistemas operativos de disco DOS.

Formato: | FILES *espefich* |

Esta orden lista todos los especificadores ficheros. Al igual que DIR, puede tener un *espefich,* pero ha de estar encerrado entre comillas. Al igual que en DOS, los dos caracteres *comodines* ? y * están disponibles.

Ejemplos:

`FILES "*.BAS"`	*Lista todos los especificadores de ficheros con la extensión BAS, de la unidad por defecto, A.*
`FILES "B:*.BAS"`	*Lista todos los ficheros de la unidad B con la extensión BAS.*
`FILES "B:*.*"`	*Lista todos los ficheros de la unidad B con independencia de su nombre o extensión.*

```
FILS "CAPI??.MAC"
```

Lista todos los ficheros de la unidad por defecto, cuyo nombre comienza por CAPI y va seguido de uno o dos caracteres y cuya extensión es MAC.

5.3. KILL. Eliminación de ficheros

La orden KILL se utiliza para eliminar o borrar un fichero del disquete.

Formato: | KILL *espefich* | *Ejemplo KILL "B:MOTOR"*

Precaución: Antes de ejecutar esta orden, asegúrese de que realmente desea suprimir ese fichero de su disquete.

5.4. NAME. Cambio del nombre de un fichero

Cambia el nombre de un fichero. Es similar a la orden RENAME en DOS.

Formato: | NAME *espefich* AS *nombrefich* |

El fichero denominado *espefich* se cambia por el nuevo nombre *nombrefich.*

Ejemplo:

```
NAME "B:DEMO4.MAC" AS "DEMO5.MAC"
```

El fichero de la unidad B que se llamaba DEMO4.MAC se llamará ahora DEMO5.MAC.

Precaución:

Es preciso que *espefich* exista y *nombrefich* no existe en el disco; en caso contrario se producirá un error. Obsérvese que en el nuevo nombre no incluye unidad, ya que el nuevo fichero permanece en el mismo disco y en la misma área del disco.

5.5. MERGE. Fusión de programas

LOAD y RUN siempre borran el programa actualmente en memoria antes de cargar un nuevo programa. La orden MERGE carga el programa desde el fichero especificado y lo fusiona o mezcla con el programa existente en memoria, formando un programa más grande.

Formato: | MERGE *espefich* |

Esta orden permite escribir programas en pequeños segmentos y más tarde ponerlos todos juntos con MERGE para hacer un programa más grande. Esta técnica permite comprobar y depurar pequeños programas, que siempre serán más manejables que los grandes programas.

Suponga que tiene un programa DEMO1 en memoria 1, otro DEMO2 en el disco de la unidad A. Cuando ejecuta

```
MERGE "DEMO2"
```

cada línea de este programa se inserta en su propia posición de acuerdo a su número de línea, en el programa DEMO1. Si una línea cargada desde el fichero tiene el mismo número de línea que una línea de DEMO1, la línea de DEMO2 sustituye a la que estaba en memoria. Por consiguiente, si desea fusionar programas, deberá tener la precaución de que no coincidan los números de línea.

Una observación importante es que los programas deben *ser conservados en disco como ficheros ASCII.* Recuerde que los programas BASIC pueden quedarse en formato binario comprimido si utiliza la orden SAVE *«nombre de fichero»* y que para conservar un programa en ASCII —tal como aparece en pantalla— debe utilizar el formato SAVE «nombre de fichero», A.

Ejemplo:

Supongamos que tiene un programa en memoria (cuyas líneas van desde 10 hasta 500) y que posteriormente desea mezclarlo con otro u otros nuevos que va a escribir. El nombre es TEST1, y para conservarlo en la unidad de disco, ejecute

```
SAVE "TEST1.BAS",A
```

Borre su programa de memoria con la orden

```
NEW
```

Escriba su nuevo programa TEST2 (con líneas desde 600 hasta 1000) si desea mezclarlo con el anterior. Ejecute

```
MERGE "TEST2"
```

ahora tendrá un nuevo programa cuyas líneas van de 10 a 1000 y que contiene los dos programas TEST1 y TEST2. A partir de ahora ya podrá procesar este programa como una nueva unidad completa.

NOTAS:

1. Si el programa TEST1 no se hubiera conservado en formato ASCII, resultará el mensaje de error «Bad file mode».

2. Si TEST1 y TEST2 tuviesen líneas comunes, se borrarán tras MERGE las líneas de igual número de TEST1 y el nuevo programa tendrá las nuevas líneas de TEST2.

5.6. CHAIN. Encadenamiento de programas

CHAIN transfiere el control a otro programa, y al mismo tiempo pasa los valores de las variables a él, desde el programa actual. Este mecanismo es útil para correr programas muy grandes que no se ajustan al espacio de memoria; para ello se rompe el programa grande en varios más pequeños y se corren cada uno de ellos independientemente.

Formato: CHAIN *espefich*

`CHAIN "PRUEBA 2"` *Carga y ejecuta el programa PRUEBA 2*

La ejecución de PRUEBA 2 comienza en la primera línea del programa.

```
1000 CHAIN "SEGPROG"
```

Esta orden se puede añadir al final del programa para pasar el control al segundo.

Variantes del formato CHAIN

1.

CHAIN *espefich,línea*

`CHAIN "PRUEBA2",500` *Ejecuta el programa «PRUEBA2» comenzando en la línea 500*

`CHAIN "PRUEBA2",100` *Ejecuta el programa «PRUEBA2» comenzando en la línea 1000*

Esta opción del número de línea es interesante, sobre todo si se encadena a un programa que ya se ha ejecutado al menos una vez.

2. Para pasar variables del programa actual al programa encadenado, se debe especificar qué variables se han de pasar.

CHAIN *espefich, línea,*ALL

```
CHAIN "PRUEBA2",100,ALL
CHAIN "PRUEBA2",,ALL
```
Carga todas las variables

3. Con frecuencia sólo se necesita pasar alguna de las variables a otro programa. En ese caso debe incluir una sentencia COMMON en el programa actual (encadenante) para transferir variables al programa encadenado.

```
COMMON TEXTOS$(),LIBRE,ACTUAL,ULTIMOS
```

COMMON lista las variables que se van a transferir, y se sitúa normalmente cerca del principio del programa.

Para indicar una variable de subíndice, de índice o de matriz, su nombre debe ir seguido con (:)

```
COMMON TEXTOS$(),LIBRE,NOMBRE,NOTAS
```

Después de que se ha ejecutado esta sentencia:

```
CHAIN "DEMO"
```

o bien

```
CHAIN "DEMO",100
```

transfieren las variables TEXTO$,LIBRE,NOMBRE y NOTAS al programa DEMO.

4. El programa cargado por CHAIN se puede fundir con el programa actual en memoria, de modo que el programa cargado sustituye solamente parte del programa existente en lugar del programa completo. Un programa que sustituye parte de un programa existente se llama *recubrimiento.*

```
CHAIN MERGE "DEMO2"
```

funde el programa DEMO2 (conservado en formato ASCII) con el programa actual en memoria.

5. El uso de DELETE permite borrar un recubrimiento

```
CHAIN MERGE "DEMO2",DELETE 500-1000
```

Las líneas 500 a 1000 se borrarán del programa actual antes de que DEMO2 sea cargado.

NOTAS:

1. La sentencia CHAIN deja los ficheros abiertos.
2. Si se omite la opción MERGE, los ficheros se cierran antes de cargar el programa encadenado.
3. La sentencia CHAIN con MERGE preserva la carga actual de OPTION BASE.
4. Si se omite MERGE, no se preservan los tipos de variables o funciones definidas por el usuario para ser utilizadas por el programa encadenado. Es decir, cualquier función DEFINIT, DEFSNG, DEFDBL, DEFSTR o DEF FN que contengan variables compartidas debe ser redefinida en el programa encadenado.

6. LOS CARACTERES DEL TECLADO

Los lenguajes de programación, y el BASIC no es una excepción, se definen mediante un alfabeto a partir del cual se constituyen palabras (instrucciones o datos) que constituyen la gramática del lenguaje y como los lenguajes humanos se rigen por unas normas que será preciso conocer.

Los actuales teclados de las computadoras personales comprenden todos los caracteres utilizables en los diversos lenguajes. Estos teclados suelen ser mucho más completos que sus equivalentes en las máquinas de escribir. Los tres modelos estándar que dotan a cualquier computadora son: QWERTY, AZERTY y Dvorak.

Teclado QWERTY. Es el teclado por excelencia y el que dota a la mayoría de las máquinas que se venden en el mundo, incluido naturalmente España. Se basa en que la disposición de las teclas es similar al de las máquinas de escribir, y las seis teclas alfabéticas de la segunda fila de teclas son Q-W-E-R-T-Y.

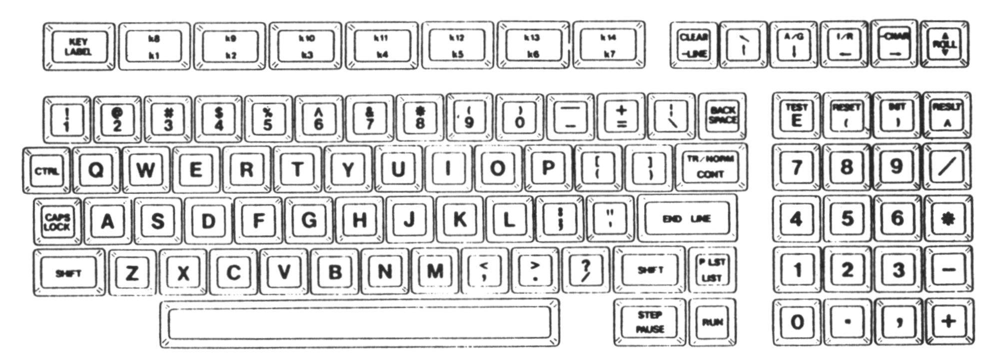


Figura 1.16. Teclado QWERTY de la computadora HP-86B.

Teclado AZERTY. Es un teclado similar al QWERTY pero que sólo se utiliza en Francia o en los países francófonos.

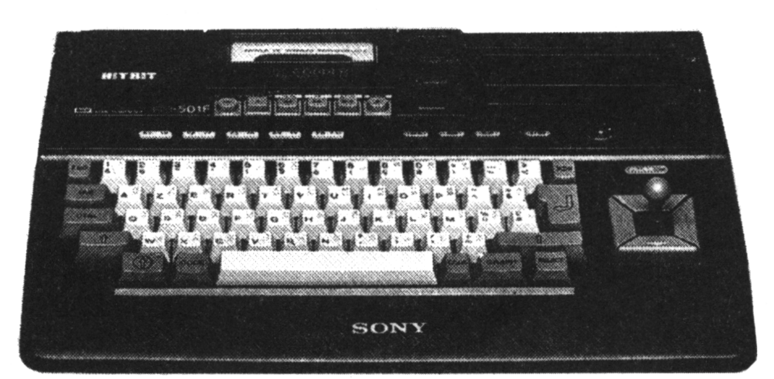


Figura 1.17. Teclado AZERTY de la computadora SONY MSX.

Teclado Dvorak. Nuevo teclado que comienza a instalarse en computadoras procedentes de Estados Unidos (el modelo Apple IIc, del gigante Apple, es buena prueba de ello). Al decir de sus diseñadores y seguidores, es el teclado más *ergonómico* (cómodo al usuario) que existe en el mercado y el teclado del futuro.

Por ahora, en Europa, se sigue trabajando fundamentalmente en QWERTY con la excepción parcial de Francia.

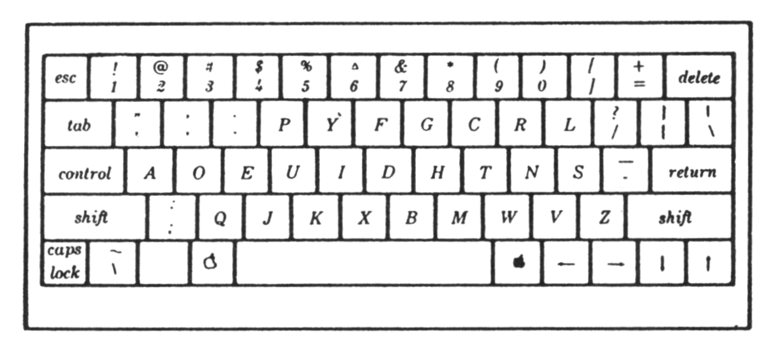


Figura 1.18. Teclado Dvorak de la computadora Apple II c.

6.1. Juego de caracteres

El juego (conjunto) de caracteres de una computadora puede incluir: caracteres alfanuméricos (letras y números), caracteres especiales y caracteres gráficos. Con la excepción de los caracteres estándar de las máquinas de escribir, los caracteres especiales y gráficos variarán de un modelo de computadora a otra.

- *Caracteres alfabéticos:* letras mayúsculas y minúsculas (A a Z).
- *Caracteres numéricos:* 0 a 9.
- *Caracteres especiales:*

Carácter en blanco, representado en impresión por ⊔,
= *Igual u operador de asignación.*
- *Signo menos.*
+ *Signo más.*
* *Operador de multiplicación arimética o símbolo de asterisco.*
/ *Barra inclinada u operador de división.*
∧ o ↑ *Operador de exponenciación (algunas versiones de BASIC representan la exponenciación mediante* ***).*
% *Símbolo de porcentaje o tanto por ciento. Identificador de variable entera.*
Cardinal. Símbolo utilizado en tratamiento de ficheros.
(*Paréntesis a la izquierda.*
) *Paréntesis a la derecha.*
$ *Signo dólar. Símbolo utilizado para representar variables de cadena.*
? *Signo de interrogación. (Abreviatura de sentencia PRINT en algunas versiones de BASIC).*
& *Signo «Y» de suma (comercial).*
! *Punto de exclamación. En algunas versiones de BASIC es equivalente a sentencia REM.*
, *Signo coma (separación de mensajes en sentencias PRINT).*
; *Signo punto y coma.*
[*Corchete a la izquierda.*
] *Corchete a la derecha.*
" *Símbolo de comillas.*
< *Símbolo de «menor que».*
> *Símbolo de «mayor que».*
@ *«a» comercial.*
— *Subrayado.*
⇆ *Símbolos utilizados normalmente para indicar los movimientos del cursor.*

Además de los caracteres anteriores, en el teclado suelen existir teclas de función, de control, etc., como se explicó en el párrafo 4.1 del Capítulo 1.

- *Caracteres gráficos*

Existen algunas computadoras (Commodore, entre otras marcas) que poseen una gran cantidad de caracteres gráficos que suelen utilizar especialmente para representaciones gráficas, juegos, etc.

EJERCICIOS DE REVISION

1. Lea los capítulos de introducción del *Manual de Usuario y Programación* de su computadora y practique todo lo aprendido en esta lección con la misma. No se preocupe si posee alguna tecla u orden distinta de las descritas en la lección; con toda seguridad será similar a alguna de las tratadas, si no fuera así, lea detenidamente su descripción.
   Compruebe el correcto funcionamiento de todas las teclas y realice operaciones aritméticas con su computadora.

2. Introduzca el siguiente programa en memoria. (No olvide pulsar la tecla RETURN al final de cada línea.)

```
10 REM DEMO
15 REM
20 A=5
30 B=6
40 C=A*B-4
50 PRINT "EL RESULTADO DE LA OPERACION ";
60 PRINT "ES";C
70 PRINT "FIN"
75 PRINT
80 END
```

— Realice las siguientes operaciones:

a) LISTAR el programa completo.
b) LISTAR la línea 40.
c) LISTAR las líneas 50 a 70.
d) Insertar la línea 35 D=B/2.
e) Borrar las líneas 15 y 75.
f) Modificar la línea 50 de modo que resulte:

```
50 PRINT "EL RESULTADO DE LA OPERACION "
```

g) Ejecute el programa: RUN, RUN 60.
h) Introduzca la línea 55 PAUSE o 55 STOP.
i) Ejecútelo de nuevo.
j) Continúe su ejecución tras la interrupción.
k) Conserve el programa en su unidad de casete o disco con el nombre DEMOX.
l) Cargue el programa DEMOX

- Con la orden LOAD "DEMOX"
- Con la orden RUN "DEMOX" (si posee unidad de disco).

CAPITULO 2

Elementos del lenguaje de programación

0. INTRODUCCION

Tras estudiar este tema aprenderá:

— Los elementos fundamentales del programa: sentencias, datos, etc.
— Diferentes tipos de datos utilizados en los programas.
— El tratamiento de los datos.
— El tratamiento de un programa: introducción, ejecución, listado, etc.
— El concepto de operador y su tratamiento.
— Los operadores aritméticos y lógicos.
— La realización de expresiones aritméticas.
— El concepto de variable y su definición.

1. ELEMENTOS DE UN PROGRAMA

Un programa, como ya se ha definido, *es un conjunto de líneas numeradas en orden creciente de modo que contienen sentencias y órdenes* que instruyen a la computadora sobre las operaciones que ha de realizar.

Las versiones de BASIC suelen permitir números de línea comprendidas entre 0 y 65535. El número máximo de líneas BASIC permitidas en un programa puede variar de un modelo a otro. Los números de línea determinan la secuencia de instrucciones en un programa: la primera línea debe tener el número de línea más pequeño y la última línea debe tener el número de línea más grande.

Un programa sencillo es:

```
10 INPUT C
20 F=32+1.8*C
30 R=0.8*C
40 PRINT F;R
50 INPUT A$
60 IF A$="SI"  THEN  10
70 END
```

En diversas versiones de BASIC se admiten más de una sentencia en una línea siempre que en ellas exista un separador: (:) en Commodore/APPLE II/TRS-80/VIC-20, etc.; (@) en HP-85, etc.

Cuando se ejecuta un programa en BASIC se tratan las sentencias una a una, comenzando con la primera y prosiguiendo hasta la última. Algunas sentencias, tales como GOTO, IF...T-HEN, GOSUB, cambian esta secuencia ascendente.

1.1. Sentencia

Una sentencia es una instrucción completa en BASIC que instruye a la computadora para ejecutar diferentes operaciones.

Por ejemplo:

```
GOTO 400
```

Instruye a la computadora para que vaya a ejecutar las sentencias existentes en la línea 400 y siguientes.

```
PRINT "BUENOS DIAS"
```

En esta ocasión los datos que se tratan son alfanuméricos y permiten imprimir en la pantalla BUENOS DIAS.

1.2. Ordenes (comandos)

Son determinados mandatos que se ejecutan directamente en el momento que el usuario lo desea, bien en modo directo o en modo programa y que permiten ejecutar, listar o depurar un programa así como el tratamiento de ficheros y gráficos.

Son órdenes: RUN, LIST, LOAD, SAVE, etc.

1.3. Expresiones

Una expresión es un conjunto de datos. Existen dos tipos de expresiones:

1. *Expresiones numéricas.* Están compuestas de datos numéricos.

   Ejemplos:

```
(1+4*3)/5                          A
6*B                                4.7654
ABS (X)+RND (1)                    SIN(3+F)
```

2. *Expresiones de cadena.* Están compuestas de datos alfanuméricos.

   Ejemplos:

```
A$="LUIS GARCIA GARCIA"            B$
"BUENOS DIAS"                      A$+B$+C$
```

1.4. Funciones

Las funciones son subrutinas integradas de modo automático en BASIC que permiten ejecutar cálculos con datos. Se pueden utilizar funciones de igual modo que se utilizan datos como parte de una sentencia.

Algunas funciones en BASIC son:

```
INT,MAX,SIN(X),LEFT$,LEN...
```

1.5. Operadores

Son símbolos aritméticos o palabras reservadas que realizan operaciones aritméticas, lógicas y de relación.

Ejemplos:

```
7+5*8
6 AND 14
9<25
```

1.6. Pruebas (test)

BASIC ejecutará dos clases de pruebas (test) para ver si existe una relación entre dos o más operaciones:

1. Pruebas de relación, que comprueban la relación de equivalencia entre las dos expresiones.

   Ejemplos:

   ```
   A=5
   A$<B$
   A$=<C$
   ```

2. Pruebas lógicas, que comprueban la relación lógica entre expresiones.

   Ejemplos:

   ```
   A$="SI" AND B$="NO"
   C<6 OR M<4 OR P>5
   ```

2. TIPOS DE DATOS

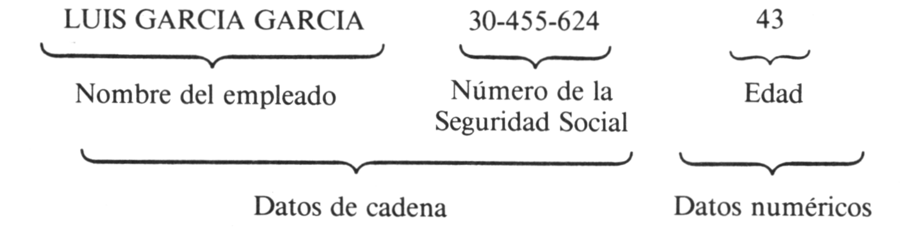
La gestión de una computadora en lo relativo a los datos a tratar por la misma comprende tres fases: introducción, manipulación y extracción de datos.

Las computadoras funcionan con muchas clases de información, y a la misma se le denomina de modo general *datos.* Los datos pueden consistir en números o palabras. A los números se les conoce como *datos numéricos* y a las palabras como *datos alfanuméricos* o *cadenas.*

Consideramos el caso de un empleado de una empresa de comestibles cuya ficha de la Seguridad Social es:

```
LUIS GARCIA GARCIA       43       30-455-624
```

Así, «LUIS GARCIA GARCIA» es un grupo de palabras que la computadora trata como un texto único; «43» es un dato numérico; y «30-455-624», ¿es un texto o un dato numérico? Los dos guiones se clasifican por la computadora como datos de texto o alfanuméricos, por consiguiente el número de la Seguridad Social se debe tratar como una «cadena».



2.1. Datos numéricos

Los números, en las microcomputadoras, se pueden representar en los sistemas de numeración decimal, octal y hexadecimal.

Existen dos clases de números que se pueden almacenar en la memoria de la computadora: *enteros y reales.*

Existen cuatro tipos de números que BASIC reconoce. Estos son: enteros, reales, hexadecimales y octales.

2.1.1. Números enteros

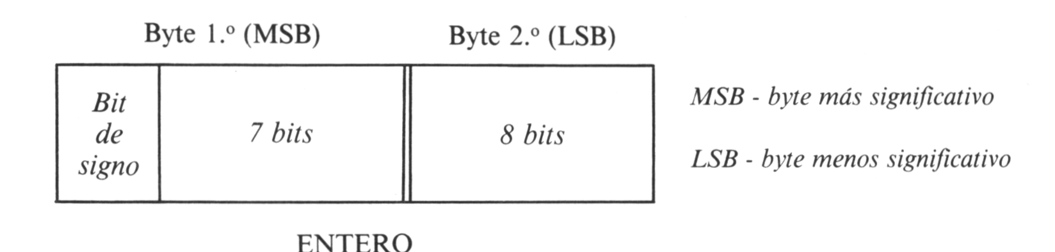
Un entero es un número que no tiene parte decimal o punto decimal. El número puede ser negativo si va precedido de un signo (−) o positivo si va precedido de un signo (+). Un número sin signo se considera positivo.

Los signos coma no se permiten en estos números.

El rango de números enteros admitidos por BASIC es −32768 a +32767 ambos inclusive. Los siguientes ejemplos muestran enteros válidos y no válidos:

-32	Válido
4	Válido
15.6	No válido (posee parte decimal)
4003	Válido
45840	No válido (mayor de 32767)
-33000	No válido (menor de −32768)

En su PC se necesitan dos bytes (dieciséis bits) para representar cada número entero. Mediante dos bytes se puede representar cualquier entero desde 0 a 65535 (2^{16}=65536). Sin embargo, a fin de utilizar los números negativos, de los 16 bits uno se debe utilizar para el signo, por lo que quedarán sólo 15 bits para representar números, o sea 2^{15}=32768, y el rango −32768 a +32768 con el cero en el punto central.



ENTERO

2.1.2. Números reales

Los números reales pueden ser enteros, o números con parte decimal o números con sólo parte decimal. Igual que los enteros pueden ser positivos o negativos.

Los números reales se pueden representar en BASIC en dos formatos que le dan nombre a dichos números: coma fija y coma flotante.

Números de coma fija

Los números de coma fija (o punto fijo) son números reales positivos o negativos, es decir, números que contienen puntos decimales (en BASIC no se pueden poner comas para represen-

tar números). BASIC puede operar con números de 17 dígitos pero sólo se visualizan 16. Los siguientes ejemplos muestran números de coma fija válido y no válido.

45.327	*Válido*
6,24	*No válido (coma no permitida)*
0.000001110000076541	*No válido (demasiados dígitos)*

Números de coma flotante

Los números de coma (punto) flotante son números reales representados en forma exponencial (conocida como notación científica). Un número en notación científica o de coma flotante tiene el siguiente formato:

± *número E (o D)* ± *e*

número	número entero o de coma fija
letra E o *D*	simple precisión /D, doble precisión
e	entero con signo en el rango −38 a +38

Ejemplos:

Notación estándar	Notación científica
300	3E2
.23	23E-2
-.54	-54E-2
12345678900	1.23456789E+10
.0000446979	446.979E-7
4469000000	4469D6

El rango de números de coma flotante es 1E−38 a 1E+38.

La notación científica es un método muy útil para representar números muy grandes o muy pequeños.

Cualquier número más grande que el límite superior dará un error de desbordamiento de capacidad. De igual modo un número más pequeño que el límite inferior produce un resultado cero.

```
PRINT 2.7-141285E+38        PRINT 3.88743245-39

?Overflow                   0
```

Normalmente los números enteros se convierten automáticamente en representaciones de comas flotantes antes de utilizar en cálculos aritméticos, si no han sido definidos previamente como enteros.

Los rangos de números reales dependen de las diferentes versiones de computadoras:

BASICA del IBM PC

Número menor: 1E−38
Número mayor: 10E+38

Applesoft de Apple II

Número menor: 1E−38
Número mayor: 1E+38

BASIC del Commodore 64/128

Número menor: +2.93873588E−39
Número mayor: +1.70141183E+38

NOTA 1: *Cualquier número más grande que el límite superior producirá un error de desbordamiento de capacidad. De igual modo un número más pequeño que el límite inferior produce un resultado cero.*

Unos ejemplos en el caso de la computadora Commodore 64 son los siguientes:

```
PRINT 1.60452283E+38      No existe desbordamiento
1.60452283E+38
PRINT 1.70141184E+38
OVERFLOW ERROR            Número mayor que el límite superior
PRINT 3.43874597E-39
0
```

NOTA 2: *Los números enteros se convierten automáticamente en números reales en los cálculos aritméticos. Si no se han definido previamente como enteros.*

2.1.3. Uso correcto de los números o constantes numéricas

Cabe preguntarse ante los diferentes formatos para introducir en memoria los datos numéricos ¿por qué utilizar por separado números enteros y números reales? La respuesta radica en el diferente espacio de memoria ocupado por los números enteros y los números reales. Así por ejemplo en Microsof Basic, las constantes numéricas que se introducen en formato de número entero, hexadecimal u octal se almacenan en dos octetos de memoria y se interpretan como números enteros o completos. Las constantes numéricas de simple precisión se almacenan en 4 octetos de memoria y con 7 dígitos, mientras que las constantes de doble precisión se almacenan en 8 octetos de memoria y 17 dígitos.

Redondeo

Los números reales se suelen redondear por exceso en los cálculos cuando el primer dígito sin precisión sea 5 o mayor que 5, y se redondea por defecto si dicho dígito es 4 o menor que 4.

```
PRINT .6666666668         Commodore 64
.666666669
PRINT 95400014.7693       Apple II
95400014.8
PRINT 150000478.85        IBM PC (simple precisión)
1500005
```

2.1.4. Números hexadecimales

Además del sistema binario, BASIC reconoce otros dos sistemas de numeración: sistema hexadecimal o de base dieciséis y sistema octal o de base ocho.

El sistema hexadecimal posee dieciséis dígitos, los números 0 a 9 y las letras A a F, que equivalen a los números decimales 0 a 15, de acuerdo con la siguiente tabla 2.1:

Tabla 2.1. Sistema de numeración hexadecimal.

Dígito	*Valor decimal*	*Valor binario*
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Cada dígito hexadecimal se representa con cuatro dígitos BASIC reconoce números hexadecimales de hasta cuatro dígitos de longitud. Cada dígito tiene un peso ponderado de una potencia de 16.

$n_1\ n_3\ n_2\ n_1$ en hexadecimal equivale a $n_1.16^0+n_2.16^1+n_3.16^2+n_4.16^4$

$16^0=1 \quad 16^1=16 \quad 16^2=16=256 \quad 16^3=16\times16=4096$

Los números hexadecimales deben estar precedidos con el prefijo &H. Ejemplos de números hexadecimales válidos son:

&H5 equivale a 5 en decimal
&HA74 equivale a (4×1+7×16+A(10)×256=) 2676
&H00FF equivale a (15×1+15×16+0×256+0×4096=) 255

Los números hexadecimales se utilizan normalmente cuando se trata con direcciones del mapa de memoria del PC.

2.1.5. Números octales

Los números octales pertenecen al sistema de numeración octal (base 8). Los dígitos octales son ocho, del 0 al 7. De este modo el diez binario (un cero) es el número 8, el once (un uno) es 9 en el sistema decimal, y así sucesivamente. Los números octales en BASIC pueden tener hasta seis dígitos, cada uno de ellos en el rango cero a siete. La tabla 2.2. muestra los dígitos octales con sus equivalentes decimales y binarios.

Tabla 2.2. Sistema de numeración octal.

Dígito	*Valor decimal*	*Valor binario*
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

Cada dígito octal se representa con 3 bits. Los números octales deben ir precedidos del prefijo & o bien &0. Algunos ejemplos de números octales válidos son:

&425	$(4.8^2+2.8^1+5)$ 277 en decimal
&237	$(2.8^2+3.8+7)$ 159 en decimal
&12	(1.8+2) 10 en decimal

Ejemplos de los tres sistemas de numeración:

Octal	*Decimal*	*Hexadecimal*
&O 077777	32767	&H 7FFF
&O 000001	1	&H 0001
&O 000000	0	&H 0000

CONVERSION DE NUMEROS HEXADECIMALES/OCTAL A DECIMAL

Afortunadamente, para la mayoría de los usarios el tratamiento de los sistemas de numeración hexadecimal y octal y su implícita complejidad, no es frecuente. No obstante, si así lo requiriera su trabajo —bien por trabajar con los mapas de memoria o ser educador—, BASICA posee unas funciones específicas y unas reglas de conversión entre los diferentes sistemas que le facilitarán la tarea. Las funciones HEX$ y OCT$ convierten números decimales en hexadecimales y octales. Para la conversión inversa le daremos una regla práctica.

Aunque el funcionamiento de HEX$ y OCT$ se explica en el capítulo 5 dedicado a funciones y funciones de cadena en especial, le adelantaremos una breve descripción de su comportamiento como órdenes.

Conversión decimal a hexadecimal

La función **HEX$ (n)** utilizada como orden proporciona el equivalente hexadecimal de cualquier número entero (n) dentro del rango −32768 a 65535

```
PRINT HEX$(4)                PRINT HEX$(1023)
4                            3FF
PRINT HEX$(14)               PRINT HEX$(0)
E                            0
PRINT HEX$(32)               PRINT HEX$(12.75)
20                           D
```

Obsérvese que si el número no es entero (decimal) la función HEX$ redondea a un entero y a continuación convierte el número en un número hexadecimal.

Si es negativo se utilizará el formato complemento a dos. Es decir HEX$ (n) equivale a HEX$ (65535−n).

La función **OCT$ (n)** utilizada como orden proporciona el equivalente octal de cualquier número entero (n) dentro del rango −32768 a 65535.

```
PRINT OCT$(4)                PRINT OCT$(0)
4                            0
PRINT OCT$(9)                PRINT OCT$(7.85)
11                           8
PRINT OCT$(24)               PRINT OCT$(-18)
30                           76617
```

Al igual que HEX& si el número es decimal se redondea a un entero y a continuación a hexadecimal. Si el número es negativo se utiliza la forma complementaria de dos. Es decir equivale a OCT$ (65536−n).

Conversión hexadecimal a decimal

Se puede convertir de base hexadecimal a decimal, haciendo preceder al número hexadecimal con el prefijo &H.

```
PRINT &HE                    PRINT &H3FF
14                           1023
PRINT &H20                   PRINT &HD
32                           12
```

No se puede utilizar &H con variables: &H debe estar seguido de una constante.

Conversión octal a decimal

Se puede convertir un número octal a un número decimal con el uso del símbolo & («ampersand») precediendo al número octal.

```
PRINT & 04                   PRINT & 1234
4                            668
PRINT &O 30                  PRINT & 11
24                           9
```

La letra O es opcional en este formato y por ello se puede omitir. Al igual que en el caso anterior, no se puede utilizar &O con variables.

3. TRATAMIENTO DE DATOS EN BASIC

Una de las principales ventajas de los lenguajes evolucionados es liberar al programador o usuario de la microcomputadora de la manipulación de los datos directamente en memoria. Basta con asignar un nombre, sin conocer la posición exacta en donde se situará la información, a la correspondiente constante o datos.

En este apartado proporcionamos información de cómo representar datos en BASIC y de cómo se interpretarán y almacenarán.

BASIC reconoce datos en dos formas, bien directamente, como *constantes,* o por referencia a una posición de memoria, como *variables.*

3.1. Constantes

Todos los datos se introducen en un programa como valores *constantes* y que no pueden cambiar.

Existen dos tipos de constantes: caracteres de cadenas y numéricas.

— Una *constante numérica* puede ser un número positivo o negativo.

Atención:

Los números no pueden llevar más que punto decimal (no pueden llevar comas).

4537.25 *válido* 4537,25 *no válido*

Existen seis tipos de constantes numéricas:

- Constante entera.
- Constante de coma fija: números reales positivos o negativos con parte decimal.
- Constante de coma flotante: números positivos o negativos en forma exponencial.
- Constante binaria.

- Constante octal.
- Constante hexadecimal.

Las constantes numéricas pueden ser de:

simple precisión	— 6 dígitos o menos.
doble precisión	— 7 dígitos o más.

Esta clasificación no suele ser común a todas las computadoras, como es la clasificación anterior.

— Una *constante de caracteres* se llama *cadena.* Una cadena es cualquier carácter o secuencia de caracteres comprendidos entre comillas. Cualquier carácter distinto del signo de interrogación se puede introducir en una cadena.

Ejemplos de cadenas son:

```
"BUENOS DIAS"
"ERES GRANDE ENANO"
"12 DE OCTUBRE DE 1982"
```

Ejemplos de aplicación de cadenas:

```
PRINT "HOLA MACKOY"
HOLA MACKOY
```

Dentro de una cadena se pueden incluir caracteres alfabéticos, numéricos, símbolos especiales, caracteres gráficos o caracteres de control.

En una sola cadena se pueden almacenar hasta 256 caracteres. A la cadena que no tiene caracteres se le llama *cadena vacía. Longitud de una cadena* es el número de caracteres que contiene la misma.

Existen algunos caracteres que no se pueden representar directamente en cadenas, por ejemplo las comillas. Sin embargo, cuando se estudien las funciones de cadena se verá que mediante la función CHR$ se pueden representar aquellos caracteres que no pueden serlo directamente.

3.1.1. Precisión de las constantes numéricas

En BASIC se pueden representar los números de tres modos diferentes: enteros, simple precisión y doble precisión. La precisión de los números se refiere al número de dígitos utilizados para representarlos. Si se dice por ejemplo que un número tiene 16 dígitos de precisión, sólo los 16 primeros dígitos son exactos; cualquier dígito extra sólo puede ser utilizado para redondeo. La precisión entera se utiliza para los números enteros, octales y hexadecimales; y simple y doble precisión se emplean para números de coma fija y de coma flotante.

ENTEROS - %

Los números enteros no tienen parte decimal

5		5/6	
24	*números enteros*	74.5	*números no enteros*
756		0.00275	

En su micro, dos bytes (dieciséis bits) se utilizan para representar un número entero.

SIMPLE PRECISION - !

Los números de simple precisión se representan con site dígitos de los cuales seis son precisos. Un número es de simple precisión si no está definido como entero, octal o hexadecimal y se escribe de la siguiente forma:

- Siete dígitos o menos.
- Número seguido de un signo de explicación (!).
- En formato exponencial utilizando E.

Ejemplos de números de simple precisión son:

```
325.725
32!
1.071821E+07
```

Observe el siguiente ejemplo:

```
PRINT 335.24/3
111.746
OK
```

Como es un cálculo de simple precisión se ha representado con 6 dígitos, aunque el resultado verdadero es 111.74666...

Observe que si hace A!=37 está asegurando que su PC interpretará el valor de A! como 3.700000 E+01 y no como el entero 27.

El número más grande en simple precisión es $\pm$1.701412E+38 y se almacena en 4 bytes (de 8 bits cada uno). El número más pequeño en simple precisión es $\pm$2.938735E-39.

Se necesitan cuatro bytes para representar un número de simple precisión.

DOBLE PRECISION - #

Con frecuencia es necesario tener un número expresado con más de 7 dígitos significativos. Para ello se utilizan los números de doble precisión que se almacenan con 17 dígitos de los cuales 16 son precisos y se visualizan.

Un número se define de doble precisión si se escribe con:

- Ocho dígitos o más.
- Forma exponencial, utilizando D (doble precisión).
- El signo # al final del número.

Ejemplos de números de doble precisión son:

```
745483912
-1.07324D-05
7643.1#
3456721.1234
```

Los números de doble precisión se almacenan en ocho bytes. El número más grande que se puede representar con doble precisión es $\pm$1.701411834604692D+38 y el número más pequeño es $\pm$2.938735877055719D-39.

Notación científica

10417309	*equivale a*	1.041731×10^{7} = 1.041731E+07
−0.000000010417309	*equivale a*	-1.04173×10^{-7} = −1.041731E−07
1.234.567.890.123.456.789	*equivale a*	$1.2345678901234567\times10^{18}$ 1.2345678901234567D+18

NOTA: Sin embargo, el lector se habrá preguntado la razón de esta clasificación, cuando los números en su amplia acepción se pueden considerar todos como reales y dentro de ellos enteros o decimales. La respuesta a su pregunta es que el ordenador, al reservar espacio de memoria para sus números, ocupa diferentes tamaños según sea el tipo de números. Así los números enteros ocupan en memoria 2 bytes, los números de coma flotante ocuparán 4 u 8 bytes, según sean de simple precisión o doble precisión.

3.2. Variables

Los datos a tratar, como ya se ha dicho, pueden ser constantes o variables. Así, supongamos que se desea obtener la longitud de una circunferencia de cualquier radio:

$$C=2\times\pi\times R=2\times 3.141592\times R$$

El número 3.141592 que representa a π es una constante. El radio del círculo es la variable que puede cambiar produciendo un resultado diferente. Cualquier tipo de datos se puede representar en un programa como constantes o variables.

Las *variables* representan a los datos por su nombre y no por su valor. Destaquemos, sin embargo, que esta noción de variable no tiene nada que ver con las variables e incógnitas utilizadas en las funciones y ecuaciones algebraicas.

Una variable designa simplemente una información denominada mediante un nombre. El valor no tiene por qué ser siempre el mismo; se puede cambiar el valor de la variable a lo largo del programa.

Ejemplo:

```
A=2*3.141592*5
```

A es una variable numérica cuyo valor es el resultado de 2*3.141592*.

De lo anterior se deduce que los datos de entrada se almacenan en variables. Una variable es un lugar para almacenar información en la memoria de la computadora. Las variables pueden utilizarse para almacenar datos numéricos o datos de cadena.

Pruebe el siguiente ejemplo:

```
10 NUMERO1=425
20 NUMERO2=36
30 CADENA$="Esta ciudad es demasiado grande para los dos"
40 PRINT NUMERO1-NUMERO2
50 PRINT CADENA$
60 END
```

NUMERO1 y NUMERO2 son variables numéricas y CADENA$ variable de cadena.

Al objeto de poder tratar los datos mediante variables será preciso darle un nombre a las mismas. Para elegir los nombres de las variables es necesario respetar ciertas reglas restrictivas en la elección de los mismos. La representación de las variables difieren de una versión BASIC o otra, por lo que si bien indicaremos aquí los criterios de diferentes computadoras, a lo largo del curso seguiremos un criterio único para no confundir al lector.

3.2.1. Nombres de variables numéricas

Un nombre de variable puede tener cualquier longitud (40 en MBASIC, 31 en CBASIC, etc.), pero sólo son significativos los *n* primeros caracteres (*n* es 2 en Apple II, Commodore 64, Dragon 64, etc.; *n* es 40 en BASICA del IBM PC).

Reglas de escritura

- El primer carácter del nombre debe ser una letra.
- Los caracteres permitidos en los nombres son letras, números y el punto decimal (.) como único signo de puntuación.
- Un nombre de variable no puede ser una palabra reservada (órdenes, sentencias, funciones y operadores), aunque sí puede contener palabras reservadas.

EXP	no es correcta como nombre de variable
EXPONENTE	es correcta como nombre de variable

- Un nombre de variable que comienza con FN se supone que es una llamada a una función definida por el usuario (capítulo 6).
- Se puede incluir al final del nombre un carácter específico denominado de declaración del tipo de variable que sirve para definir el tipo y precisión de la variable.

Estos caracteres son: %, !, #, que representan a las variables enteras, de simple precisión y doble precisión.

El siguiente ejemplo muestra nombres válidos y no válidos de variables numéricas.

Nombres válidos

Variables numéricas

A
B1
PRIMER.NUMERO
XY
ZACA1

Nombres no válidos

Variables numéricas

1A3
31
3DE

Al igual que sucede con las constantes numéricas, las variables numéricas se clasifican en: enteras, simple precisión y doble precisión.

VARIABLES ENTERAS

Un nombre de variable seguido por un símbolo % define una variable entera. Por ejemplo:

PRECIOS%
INTERES%
A1%

Introduzca y ejecute el siguiente programa:

```
10 A%=5: B%=6.3: C%=8.7
20 PRINT A%,B%,C%,A
```

Cada una de las variables se almacena en BASIC como un número redondeado al entero más próximo; es decir, 6.3 se redondea a 6 y 8.7 a 9. Ejecute el programa:

```
RUN
5             6             9             0
OK
```

El último valor es 0, ya que la variable A no tiene valor asignado (más adelante se insistirá en este punto). Como observará el lector, A y A% son dos variables diferentes y tienen el mismo efecto que haber utilizado letras diferentes ya que es el prefijo quien declara la variable.

VARIABLES DE SIMPLE PRECISION

Las variables de simple precisión están seguidas por el símbolo ! o no llevan prefijo. BASIC supone que las variables numéricas son siempre de simple precisión si no se las declara previamente o de doble precisión. Por ejemplo, son nombres válidos de simple precisión:

NOTAS!
CALIFICACIONES
DEMO1!

Introduzca los siguientes ejemplos:

```
A!=4.814
PRINT A!
 4.814
PRINT A
 4.814
```

En este caso A y A! son la misma variable, igual que NOTAS y NOTAS!

VARIABLES DE DOBLE PRECISION

Los nombres de variables terminados con el carácter # definen a la variable como de doble precisión. Por ejemplo:

PI#
PRECIO#

NOTA: Aunque la tendencia actual es en todas las versiones BASIC disponer de los tres tipos de variables enteras y simple/doble precisión, existen computadoras que sólo manipulan dos tipos de variables: enteras y reales, como es el caso de Apple II, Commodore 64, etc.

3.2.2. Variables de cadena

Las variables de cadena representan cadenas o series de caracteres de cualquier longitud entre 0 y 255 caracteres.

Los nombres de las variables de cadena deben terminar con un signo dólar ($) y su representación es la misma que la variable numérica añadiéndole el carácter $.

Ejemplos:

```
NOMBRE$
A1$
B5$
BA$
```

NOTA: Si una variable no termina por el carácter ($) o el carácter (%), se supone que el dato representado es un número real.

Así, A, A$, A%, A! y A# son tres variables distintas:

A	*Variable numérica real*
A$	*Variable de cadena*
A%	*Variable entera*
A¡	*Variable real de simple precisión*
A#	*Variable real de doble precisión*

3.3. Reglas prácticas de manipulación de variables

¿Qué razón existe para especificar el tipo de una variable numérica?

BASIC, como se ha comentado, supone que todas las variables son de simple precisión a menos que se indique lo contrario. Ahora bien, cada variable de simple precisión ocupa 4 bytes de memoria; por consiguiente, si usted conoce de antemano que todos los números que se van a procesar en su programa son enteros (en el rango permitido) ¿por qué no declararlo así en todos ellos? Cada número supondrá un ahorro de 2 bytes; figúrese el espacio de memoria que se puede ahorrar. Por otra parte, su programa se correrá mucho más rápido, BASIC hace su cálculo más rápido con enteros que con números de coma flotante. Así pues, queda contestada la pregunta.

Si utiliza variables numéricas enteras en un lugar de simple o doble precisión (naturalmente, si su programa lo permite) se obtienen dos ventajas: 1) el programa es más rápido; 2) el programa ocupará menos espacio en memoria. Posteriormente veremos un procedimiento para declarar los tipos de variable dentro de un programa con funciones DEF.

Veamos ahora algunas reglas prácticas para el uso de palabras reservadas como variables, asignación de valores a variables y conversiones de constantes y variables.

PALABRAS CLAVES O RESERVADAS

Se conocen por *palabras claves o reservadas* todas las palabras que definen una sentencia BASIC. Cuando se ejecutan programas en BASIC, se deben *evitar las palabras reservadas como nombres de variables* ya que en este caso se podrían interpretar como sentencias BASIC con el siguiente perjuicio al programa.

ASIGNACION DE VALORES A VARIABLES

Hasta que se asigna un valor a una variable, tendrá valor cero si es numérica y nulo o valor «vacío» si es una cadena. Por supuesto los programas asignarán siempre valores enteros a las variables enteras, números reales a las variables reales y cadenas a variables de cadenas. Si se asignan datos de un tipo a una variable de otro tipo, existen versiones de BASIC que permiten mezclar los datos, mientras que otras versiones originan errores.

La asignación de una constante entera a una variable numérica es aceptable debido a que los números enteros están comprendidos en los números reales. Si se asigna un número real a una variable entera y el número real está dentro del rango posible de enteros (−32768 a 32.768), se elimina la parte decimal cuya operación se conoce por *truncación.*

Si se intenta asignar una cadena de caracteres a una variable numérica de cualquier tipo, se visualizará un mensaje de error. Por el contrario, si se asigna un entero real a una variable de cadena, se acepta el número como una cadena numérica. En otras palabras, *los dígitos numéricos se tratan como caracteres en vez de como componentes de un número.*

CONVERSION REALES/ENTEROS

Número real a entero

BASIC trunca (ignora) la parte decimal del valor original. El valor truncado debe estar entre −32768 y 32767.

Ejemplos:

```
A%=-42.5
```

asigna el valor −42 a A %

```
A%=4724.35
```

asigna el valor 4724 a A %

```
A%=2.5E+3
```

asigna el valor 2.500 a A %

```
A%=-123.45678
```

asigna el valor −123 a A %

```
A%=42.76712E3
```

asigna el valor 42.767 a A %

```
A%=42.76766E3
```

se devuelve un mensaje de error «Overflow», dado que 42768 (valor redondeado) está fuera del rango permitido de los enteros (−32767 a 32768)

Entero a número real

En la conversión de enteros a números reales el valor convertido es igual al valor original; sin embargo, ocupa 3 o 4 veces más espacio de memoria. (En MICROSOFT los enteros se almacenan en 2 octetos y los números reales en 4 u 8 octetos según sean de simple o doble precisión.)

OTRAS CONVERSIONES

En general cuando se asigna un número de una precisión a una variable con una menor precisión, el número se redondea para que concuerden la precisión implicada en el nombre de precisión más baja. Por ejemplo:

```
A=66.9037753
```

asigna 66.90378 a la variable de simple precisión A

Cuando se trata de combinar constantes y variables de diferentes precisiones, todas las variables y constantes se tratarán como si tuvieran la precisión de la variable o constante de precisión más alta. Por ejemplo:

```
A%=5: B#=4.34
PRINT A%/B#
 1.152073692213711
OK
```

En la sentencia PRINT la precisión más alta viene representada por la doble precisión de la variable B#. Como resultado, el programa utiliza la variable entera como si fuera un número de doble precisión. El resultado se visualiza como un número de doble precisión (con 16 dígitos, aunque internamente el PC le asigna 17 dígitos).

Conversiones ilegales

BASIC no puede convertir automáticamente valores numéricos a cadenas o viceversa.
Las sentencias siguientes son ilegales:

```
A$=1234
A%="1234"
```

(Se verá al estudiar las funciones de cadena que mediante VAL y STR$ se podrán realizar las conversiones. Capítulo 5.)

Definición de los tipos de variables

Además de los caracteres de declaración (%, !, #, $), ya vistos anteriormente, para especificar el tipo o clase de variable, se pueden utilizar sentencias DEF para el mismo propósito. Existen cuatro diferentes sentencias DEF:

DEFINT	*Define variables enteras.*
DEFSNG	*Define variables de simple precisión.*
DEFDBL	*Define variables de doble precisión.*
DEFSTR	*Define variables de cadena.*

Estas sentencias van seguidas por una o varias series de letras. Estas series de letras indican que todas las variables que comienzan con esas letras serán del tipo declarado por la sentencia.

DEFINT H	*La variable H será entera.*
DEFINT A-L	*Las variables A a L (inclusive) son variables enteras.*
DEFSNG H-K,R	*Las variables A a H y R se declaran de simple precisión.*
DEFDBL P,X-Z	*Declara las variables P y de X a Z de doble precisión.*

Si no se encuentra en un programa de sentencias de declaración tipo DEF, se supone que todas las variables sin caracteres de declaración son variables de simple precisión.

Las variables de cadena se pueden declarar con igual formato que las variables numéricas.

DEFSTR A-B	*Declara variables de cadena todas aquellas que comiencen por las letras A y B*

Sin embargo, le aconsejamos que utilice siempre que le sea posible el carácter declarativo $ para asignar cadenas a variables, ya que en este caso será más fácil de leer y ver qué variables son de cadena y cuáles numéricas.

4. REVISION DEL TRATAMIENTO DE UN PROGRAMA

Un programa, como ya se ha visto, es un conjunto de números de líneas, cada una de las cuales puede tener una o varias sentencias.

Así, un programa podría ser:

```
10 REM DEMO
20 REM MEDIA DE VALORES
30 A=5
40 B=8
50 C=3
60 D=6
70 PRINT "VALORES";A;B;C;D
80 PRINT "MEDIA"; (A+B+C+D)/4
100 END
```

INTRODUCCION EN MEMORIA

La forma de introducir el programa en la memoria de la computadora sería escribir mediante el teclado cada línea completa, terminando la misma pulsando la tecla RETURN o su equivalente. Recuerde antes de pulsar la tecla RETURN y asegúrese de que no hay errores de sintaxis o cualquier otro apreciable a simple vista; en este caso, corríjalos mediante teclas de edición antes de la pulsación de RETURN.

El procedimiento sería:

```
10 REM DEMO                    <RETURN>
20 REM MEDIA DE VALORES        <RETURN>
 .
 .
 .
 .
100 END                        <RETURN>
```

Una vez que se ha introducido el programa en memoria, el paso siguiente será proceder a su ejecución al objeto de comprobar su correcto funcionamiento. Se conoce como *depuración del programa* la fase de corrección o puesta a punto del mismo. En esta fase se corrigen todos los errores que posea, bien de sintaxis o bien de programación.

Para *ejecutar* («rodar», «correr» o «lanzar») un programa es preciso activar la orden RUN seguida de RETURN (teclear RUN y pulsar RETURN, o bien pulsar directamente la tecla RUN en aquellas computadoras que la posean en su teclado).

Al finalizar la orden RUN, la computadora comienza a desarrollar el programa introducido en la memoria y precisamente por la línea de número más bajo, en orden ascendente y secuencial.

Al comenzar la ejecución del programa, todas las variables numéricas toman el valor cero y las variables de cadena el valor cadena vacía, y se restauran todos los punteros de las sentencias DATA que existan en el programa.

La presentación de los resultados del programa definido anteriormente tras su ejecución (teclear RUN y pulsar la tecla RETURN) será la siguiente:

```
VALORES 5  8  3  6
MEDIA 5.5
```

Puede ocurrir que existan varios programas almacenados simultáneamente en la memoria, o bien que se desee ejecutar el programa por partes a fin de probar la «bondad» del mismo, permitiendo su depuración parcial. Para ejecutar esta operación bastará con escribir RUN seguido del número de línea con el que se desea iniciar la ejecución. Por ejemplo:

```
RUN 50       Comienza la ejecución en línea 50
RUN 100      Comienza la ejecución en línea 100
```

Es preciso notar que este método de trabajo exige conocer el programa a fondo, al objeto de comenzar en un punto que no permita falsear datos y no comencemos en una línea de programa que no exista.

En nuestro ejemplo, RUN, RUN 10 y RUN 20 ejecutan el programa completamente, dado que las líneas 10 y 20 son sentencias comentarios y no se ejecutan por el programa, existiendo sólo a efectos de documentación.

Si se realizara RUN 50, el programa comenzaría en la línea 50, con lo que los restantes anteriores a ellas no se realizarían y por consiguiente A=0 y B=0, obteniéndose el resultado:

```
VALORES 0  0  3  6
MEDIA 2.25
```

El empleo normal del comando RUN *n* (*n*, número de línea) es en los casos de realización de un programa largo y complejo, dado que suele ser buena práctica, en ocasiones, el ir probando y depurando el programa por partes a fin de ir eliminando los errores de modo parcial.

Existen computadoras que tienen la posibilidad de almacenar varios programas en la memoria central y que éstos sean llamados por su nombre para su ejecución de la forma siguiente:

```
RUN "CONTABILIDAD"
RUN "NOTAS"
```

Durante la ejecución del programa se suele insertar con frecuencia números de líneas, modificar líneas, etc. Para ello se necesita listar el programa total o parcialmente, dado que su lectura facilitará considerablemente la depuración del programa.

Un listado de un programa es la relación escrita de todas o parte de las líneas del mismo, y que aparecen en pantalla, o se imprimen en impresora para su conservación permanente o bien facilitar su lectura y depuración.

Recordemos el formato del comando LIST (su ejecución será la pulsación de LIST y tecla RETURN; normalmente no suele existir la tecla LIST, por lo que será necesario escribir L-I-S-T).

LIST	*Listado de programa completo.*
	Lista la línea número X.
LIST X-	*Lista las líneas comprendidas entre el número X y el final.*
LIST -X	*Lista todas las líneas desde el principio hasta la X inclusive.*
LIST X-Y	*Lista el programa comprendido entre las líneas X.Y.*

Veamos un ejemplo del programa DEMO:

LIST *<RETURN>*

```
10 REM DEMO 1
20 REM MEDIA DE VALORES
30 A=5
40 B=8
50 C=3
60 D=6
70 PRINT "VALORES";A;B;C;D
80 PRINT "MEDIA"; (A+B+C+D)/4
100 END
```

LIST 60- *<RETURN>*

```
60 D=6
70 PRINT "VALORES";A;B;C;D
80 PRINT "MEDIA"; (A+B+C+D)/4
100 END
```

LIST 50 *<RETURN>*

```
50 C=3
```

BORRADO DE MEMORIA

El programa contenido en la memoria se debe borrar, bien porque ya se ha terminado su depuración y puesta a punto y se ha conservado en un disco, disquete o casete, o bien porque ya no se va a utilizar más por cualquier causa.

La orden NEW borra todo el programa (o programas) almacenado en ese momento, dejando libres las posiciones de memoria que ocupaban, permitiendo que sean utilizados por otros programas.

Este comando, por consiguiente, exige un cuidado especial, ya que sólo se debe ejecutar cuando se está plenamente seguro de que el programa en memoria ya no es útil.

Por otra parte, como norma de seguridad, se ha de ejecutar siempre que se comience a introducir un nuevo programa, o tras la conservación en memoria externa (casete, disquete, etc.) del programa que se halla en la memoria principal. Si no lo hace así, se mezclarían las líneas del programa antiguo con las del nuevo y se conseguiría como resultado un programa que casi con toda seguridad no coincidiría con el que se desea introducir. Así, por ejemplo, supongamos que en memoria se tiene el programa «DEMO 1» y que se va a introducir el programa «DEMO 2» sin haber efectuado NEW.

```
10 REM DEMO 2
15 PRINT "SUMA DE NUMEROS"
20 A=5
25 B=4
30 C=6
40 PRINT "LA SUMA ES";A+B+C
80 END
```

El programa que resultaría tras introducir DEMO 2 sería:

```
10 REM DEMO 2
15 PRINT "SUMA DE NUMEROS"
20 A=5
25 B=4
30 C=6
40 PRINT "LA SUMA ES":A+B+C
50 C=3
60 D=6
70 PRINT "VALORES": A; B; C; D
80 END
100 END
```

Al ejecutar el programa se obtendría la misma solución que antes en DEMO 1 más la parte correspondiente a las líneas 50, 60, 70. Resultaría:

```
LA SUMA ES 15
VALORES 5  4  3  6
```

INTERRUPCION DE UN PROGRAMA

En numerosas ocasiones se necesita interrumpir («detener» o «abortar») un programa o la presentación de un listado, bien para efectuar alguna corrección, introducir líneas, salir de un bucle infinito, etc. La orden STOP (BREAK o BRK en algunas computadoras) interrumpe la ejecución del programa en el momento que se pulsa.

Además de como orden directa, STOP se puede utilizar como sentencia en la mayoría de las versiones BASIC, con lo cual se pueda introducir en un momento determinado en el programa.

Las razones de utilizar STOP residen en:

1. Permitir una pausa para realizar operaciones externas a la computadora.
2. Examinar resultados parciales para comprobar su veracidad.
3. Averiguar el estado de las variables del programa en un instante determinado.
4. Interrumpir la ejecución del programa por bloqueo o entrada en un bucle sin fin.
5. Otras.

La orden STOP surte efecto al finalizar la ejecución de la sentencia en curso, o en el caso de que se interrumpa un listado, cuando se ha acabado de escribir por completo una línea.

Tras la pulsación de la orden STOP (o la ejecución de la sentencia STOP) suele aparecer en pantalla un mensaje similar a:

BREAK IN LINE XXXX (XXXX número de línea correspondiente a la sentencia en que se ha detenido el programa, o lo que es igual, la próxima a realizar).

CONTINUACION DE UN PROGRAMA INTERRUMPIDO POR STOP

La orden CONT (abreviatura de CONTINUE) permite la continuación de un programa que ha sido interrumpido mediante una sentencia u orden STOP.

Normalmente esta orden no funciona correctamente si se han introducido modificaciones en el programa, o se han editado nuevas líneas de programa.

BORRADO DE VALORES DE LA VARIABLE

La orden CLR borra todos los valores que hasta ese momento se hubiesen asignado a las variables del programa, poniendo a cero los punteros de la sentencias «FOR», «GOSUB» y «DATA».

CARGA DE UN PROGRAMA

Esta orden permite *cargar* (transferir) un programa (o fichero) desde la unidad de memoria auxiliar (casete, disquete), disco duro o rígido, disco «winchester», etc.).

Formato:

LOAD *"Nombre del programa".*

En algunas ocasiones es necesario indicar el número de la unidad de la que se desea cargar y que se suele indicar mediante dígitos:

LOAD *"Nombre del programa",* D.

siendo D el número identificador de la unidad.

CONSERVACION O SALVAGUARDA DE PROGRAMAS

Mediante esta orden, permite *conservar* o *salvaguardar* (almacenar) el programa (o fichero) en la memoria auxiliar de la computadora (casete, disquete, disco duro o rígido, etcétera).

Formato:

SAVE *"Nombre del programa".*

Si es necesario indicar el número de la unidad de disco empleada, D, sería:

SAVE *"Nombre del programa",* D.

5. TRATAMIENTO DE DATOS (OPERACIONES)

En esta sección se explicará cómo se puede instruir la computadora para manipular y comprobar sus datos, lo que se conoce como «tratamiento de datos». Los dos medios de que se dispone son *operadores* y *funciones*. Dejaremos las funciones para dedicarles un tema completo y nos vamos a centrar en los *operadores con variables numéricas.*

Una *expresión* es cualquier operación aritmética o numérica, series de operaciones, y pueden ser de las siguientes formas:

- Simplemente una constante numérica o cadena de caracteres.
- Simplemente una variable numérica o caracteres.
- Una combinación de constantes y variables ligadas por operadores, como A*X∧2+B+C+D.

5.1. Operadores

Un operador es un símbolo o una palabra que significa que se ha de realizar cierta acción entre uno o dos valores que se conocen como operandos.

De modo general un operador se utiliza en las formas siguientes:

— *operando 1* OPERADOR *operando 2*

— OPERADOR *operando x*

Operando 1, operando 2 y operando x pueden ser expresiones aritméticas.

Ejemplos:

8+3	*El operador (+) suma 8 y 5 para producir el resultado 13.*
-6	*El operador actúa con un único operando 6 para producir el resultado negativo 6.*

Los operadores realizan operaciones de tipo matemático o lógico sobre los operandos. Se clasifican en tres categorías de acuerdo con las operaciones que realizan:

1. Aritméticos.
2. Relacionales.
3. Lógicos o booleanos.

De acuerdo con las clases de operandos se clasifican en:

— Numéricos.
— De cadena.
— Comprobación.

5.1.1. Operadores aritméticos

Los operadores aritméticos son bastante fáciles de comprender, dado que aprendimos a sumar, restar, multiplicar y dividir en la infancia.

Los operadores aritméticos (tabla 2.1) son: (+), (−), (*), (/), (^, **, ↑).

Tabla 2.3. Operadores aritméticos.

Operador	*Operación*
−	Negación
+	Suma
−	Resta
*	Multiplicación
/	División
\	División entera
MOD	Módulo (Resto División)
^	Exponenciación

Los símbolos + y − son operadores de suma y resta.

La barra inclinada sustituye al símbolo ÷ en la división.

El asterisco representa la multiplicación en vez de (×).

La flecha ascendente (↑) o bien (^), o (**) en otras versiones de BASIC, representan la exponenciación. A lo largo del curso utilizaremos el símbolo (^) que utilizan las computadoras APPLE, HEWLETT PACKARD, IBM PC, y la mayoría de las versiones que utilizan MICROSOFT, CBASIC y otras. COMMODORE en sus modelos 64 y 128 utiliza (↑). TRS-8- emplea (**), Amstrad utiliza (↑), etc.

Un signo + o − que aparece delante de un número indica el signo del número y se conoce como *operación unaria o monaria.*

Las operaciones de negación cambian el signo de un entero o variable real, o pueden estar al principio de una constante numérica.

Las operaciones artiméticas se pueden ejecutar con constantes o variables que pueden ser enteros o reales.

NEGACION: La negación significa que se toma el valor negativo de un número cuando este número va precedido por un signo menos (−). Por ejemplo:

```
-486                -(-225)=225
```

SUMA: El signo (+) significa que el dato (u operando) a la izquierda del signo más se debe sumar al dato (u operando) a la derecha.

```
4+7         PRINT 4+7
A1-B+C      PRINT A1-B+C
D%+4        PRINT D%+4
```

RESTA: El signo (−) significa que el dato a la derecha del signo menos se ha de restar del dato a la izquierda.

```
7-2         PRINT 7-2
A-B         PRINT A-B
44-172      PRINT 44-172
```

MULTIPLICACION: El asterisco (*) significa que el dato a la derecha del asterisco se multiplica por el dato a la izquierda del asterisco.

```
100*45      PRINT 100*45
A*B         PRINT A*B
R%*C        PRINT R%*C
```

DIVISION: El símbolo (/) indica una división ordinaria. La barra inclinada significa que el dato a la izquierda de la barra se tiene que dividir por el dato a la derecha de la barra.

```
10/2        PRINT 10/2
6400/32     PRINT 6400/32
A/B         PRINT A/B
```

EXPONENCIACION: El símbolo (^) representa la exponenciación. El dato a la izquierda del símbolo significa que se eleva a la potencia indicada por el dato de la derecha.

3^2 *El resultado es 9.*
A^B *Si A tuviera el valor 5 y B el valor 2, el resultado sería 25.*

Como ya se comentó antes, algunas versiones de BASIC utilizan ↑ y ** en lugar de ^.

DIVISION ENTERA: La división entera se representa por la barra inclinada inversa (\), al contrario de la barra inclinada directa (/), que representa a la división de números reales.

La operación división entera convierte el dividendo y divisor en enteros, redondeándolos, si fuera necesario, antes de efectuar la división propiamente dicha y tras su ejecución el cociente se redondea de igual forma.

El siguiente ejemplo muestra la operación y una aplicación.

```
323.8/9=324/9=36
427.8/9=428/9=47.555555....=48

PRINT 323.8/9                PRINT 427.8/9
 36                           48
```

MODULO ARITMETICO (MOD): El operador módulo (MOD) proporciona el resto entero de una división entera. El formato de este operador es:

```
PRINT 7 MOD 4                     PRINT 26.78 MOD 6.89
 3           resto de la           6                    resto de la
             división 7/4                               división 27/7
```

ORDEN DE EVALUACION DE EXPRESIONES

Una expresión aritmética puede constar de uno o varios operadores y operandos, como la siguiente:

```
A*B-C+10^12
```

El uso normal de los operadores en expresiones múltiples es similar a las reglas de aritmética con un orden de prioridad establecido. En el caso anterior, se realiza primero la exponenciación (^), después la multiplicación, resta y, por último, la suma.

Para el cálculo de una expresión aritmética, se exploran los operadores desde la izquierda a la derecha. Unos operadores tienen *prioridad* (precedencia) sobre otros.

La computadora procesa los datos siguiendo un orden de prioridad (jerarquía) comenzando por la prioridad superior hasta la inferior. La prioridad (mayor a menor) de los operadores aritméticos se recoge en la tabla 2.4.

Tabla 2.4. Prioridad de operadores aritméticos.

Operador	*Operación*	*Ejemplo*
^*	Potenciación	A^B
−	Negación	−A
*, /	Multiplicación	A×B
	División de coma flotante	A/B
\	División entera	A\B
MOD	Módulo	A MOD B
+, − **	Suma, Resta	A+B, A−B
* *Prioridad máxima* ** *Prioridad mínima*		
Prioridad descendente * → **		

Los operadores de igual jerarquía se evalúan de izquierda a derecha.

En una expresión que incluya dos o más operadores aritméticos los operadores de prioridad más alta se evalúan antes que los operadores de menor prioridad. Este orden de evaluación se puede anular por el uso de paréntesis, de modo que cualquier operación dentro de paréntesis se ejecuta primero.

Ejemplo 1: Prioridad normal

`45.6+1.48*4.6^2-1.47`	*Cálculo de ^*
`45.6+1.48*21.16-1.47`	*Cálculo de **
`45.6+31.3168-1.47`	*Cálculo de +*
`76.9168-1.47`	*Cálculo de −*
`75.4468`	

Ejemplo 2: Prioridad normal

```
M*N/P*Q
```

Su cálculo se realiza así:

— Todos los operadores tienen igual prioridad, 2.
— Se efectúan las operaciones de izquierda a derecha.

$$M*N$$

$$\frac{M*N}{P}$$

$$\frac{M*N}{P}*Q$$

El resultado es $\frac{M*N*Q}{P}$ y no $\frac{M*N}{P*Q}$

Ejemplo 3: Anulación de prioridad mediante paréntesis.

Si las expresiones tienen paréntesis, las operaciones contenidas en ellas se calculan en primer lugar. Por ello, se puede cambiar la secuencia de evaluación o el valor de la expresión aritmética encerrando partes de expresiones dentro de paréntesis.

`4+5*2`	*da como resultado 14*
`(4+5)*2`	*da como resultado 18*
`100*5/2-1`	*da como resultado 249*
`100*(5/2-1)`	*da como resultado 150*
`100*(5/(2-1))`	*da como resultado 500*

NOTA: *La existencia de paréntesis implica la evaluación del conjunto de paréntesis más interior.*

Ejemplo 4: Conversión de expresiones aritméticas en expresiones válidas en BASIC:

$A+2B$	`A+(2*B)` *o bien* `A+2*B`
$\frac{A-B}{C}$	`(A-B)/C`
$\frac{AB}{C}$	`A*B/C`
$\frac{A+B}{C}$	`(A+B)/C`
$(A^2)^B$	`(A^2)^B` *diferente de* `A^(2^B)` *equivalente a* A^{2^B}

5.1.2. Operadores de relación (relacionales)

Los *operadores de relación o relacionales* permiten comparar variables, constantes o cualquier tipo de datos. Las comparaciones se pueden hacer entre números reales, reales y enteros o entre cadenas. *No se pueden comparar cadenas con datos numéricos.*

Los operadores de relación más usuales se muestran a continuación en la tabla 2-5.

Tabla 2.5. Operadores de relación.

Operador	*Relación*	*Expresión*
=	Igualdad	A=B
< > o > <	Distinto (no igual)	A< >B
<	Menor que	A<B
>	Mayor que	A>B
> = o = >	Mayor que o igual	A>=B
< = o = <	Menor que o igual	A<=B

El resultado de cualquier operación es «verdadero» o «falso». Si se compara un valor con otro para ver si son iguales, la relación de igualdad es o «cierta» (son iguales) o «falsa» (son distintos). El intérprete de BASIC suele asignar a las comparaciones los dos resultados posibles.

Valor −1	*se asigna a una condición «verdadera»*
Valor 0	*se asigna a una condición «falsa»*
12<3	*se asocia el valor 0 (falso)*
2<>5	*se asocia el valor −1 (verdadero)*
2<=5	*se asocia el valor −1 (verdadero)*
3<=2	*se asocia el valor 0 (falso)*

En APPLE II, «0» es condición «falsa» y «1» es condición «verdadera».

Los operadores de comparación tienen todos la misma prioridad, pero cuando se combinan con operadores aritméticos, éstos tienen mayor prioridad.

Ejemplo 1:

```
24.5+5.8<>4.25*2.1
24.5+5.8<>8.925          (Cálculo de*)
30.3     <>8.925          (Cálculo de +)
=-1 (verdad)              (Cálculo de < >)
```

Ejemplo 2:

```
5.47*2+(21.7=5.1)
5.47*2+0                  (Cálculo =)
10.94+0                   (Cálculo *)
10.94                     (Cálculo +)
```

NOTA: *Los operadores de relación se pueden usar para comparar cadenas de caracteres.*

5.1.3. Operadores lógicos

Los operadores lógicos o booleanos permiten a los programas la posibilidad de realizar operaciones lógicas así como tomar decisiones lógicas.

Estos problemas permiten comprobar relaciones múltiples. El operador lógico proporciona el siguiente resultado:

0 si la expresión es falsa
1 si la expresión es verdadera

Si se trata de una expresión, los operadores lógicos se tratan después de los operadores aritméticos y los relacionales.

Los operadores lógicos (tabla 2.6), por orden de prioridad, son:

NOT, AND, OR, XOR, IMP, EQV

Tabla 2.6. Operadores lógicos.

Operador	*Operación lógica*
NOT	Complemento lógico
AND	Conjunción («Y»)
OR	Disjunción («O»)
XOR	«O» exclusiva
IMP	Implicación
EQV	Equivalencia

Los operadores XOR, IMP, EQV no suelen existir en algunas computadoras.

Los operadores lógicos son tratados mediante unas tablas conocidas como *tablas de verdad* que recordamos a continuación. Si usted ha estudiado en alguna ocasión la teoría de conjuntos y el álgebra de Boole le resultará sencillo su tratamiento; en caso contrario, le sugerimos que antes de proseguir lea detenidamente la tabla 2.7 hasta que entienda correctamente su funcionamiento.

Los operadores lógicos tienen igual prioridad. Si varios operadores lógicos están presentes en la misma expresión, se evalúan de izquierda a derecha. Sin embargo, en una expresión aritmética, los operadores lógicos se tratan después de los operadores aritméticos y los de relación.

Tabla 2.7. Tabla de verdad de operadores lógicos.

a) *Formato binario* (operaciones lógicas numéricas)

X	NOT X
1	0
0	1

X	Y	X AND Y	X OR Y	X XOR Y	X IMP Y	X EQV Y
1	1	1	1	0	1	1
1	0	0	1	1	0	0
0	1	0	1	1	1	0
0	0	0	0	0	1	1

b) *Formato alfabético* (operaciones lógicas de condición)

NOT

X	NOT X
V	F
F	V

X	Y	X AND Y	X OR Y	X XOR Y	X IMP Y	X EQV Y
V	V	V	V	F	V	V
V	F	F	V	V	F	F
F	V	F	V	V	V	F
F	F	F	F	F	V	F

V = Verdadero (1)
F = Falso (0)

NOTAS:

1. La operación AND da como resultado un «1», sólo si los dos bits son 1.
2. La operación OR da como resultado un «1», si uno u otro bit son 1.
3. La operación NOR complementa cada bit.
4. Para tratar los operadores lógicos se convierten A y B en enteros binarios con signo y la operación se ejecuta bit a bit, o sea el bit del resultado depende de los dos bits correspondientes de los operadores.
5. Se pueden utilizar los operadores lógicos para comprobar una cierta configuración binaria en un byte u octeto. Así, el operador AND se puede utilizar para enmascarar todos los bits salvo uno de un octeto de estado.

El operador OR permite fusionar dos octetos para crear un valor binario particular.
Se recuerda que un octeto son ocho dígitos binarios o bits.

Ejemplos:

```
    63             111111
    17             010001
---------------------------
  63 AND 17        010001      Equivalente a 17

    15               1111
    14               1110
---------------------------
  15 AND 14          1110      Equivalente a 14

    25              11001
     8              01000
---------------------------
  25 AND 8          01000      Equivalente a 8

     4               0100
     2               0010
---------------------------
   4 OR 2             110      Equivalente a 6

    43             101011
---------------------------
   NOT 43          010100      Equivalente a 20
```

PRIORIDAD TOTAL DE OPERADORES

Normalmente, las excepciones en BASIC suelen incluir operadores de diferentes tipos. Se suelen mezclar libremente los tipos en una expresión, excepto que las cadenas no pueden formar parte de las expresiones entera, reales o booleanas (lógicas). Las cadenas sólo pueden aparecer en las expresiones de cadena o de relación.

Correcto	*Incorrecto*
`4.234*(A+B)`	`4234+"BUENOS DIAS"`
`A$=B$`	`A$=B%`
`(A=B):(A$=B$)`	`NOT (M$)=N$`

Sin embargo, cuando concurren varios operadores en una expresión es preciso seguir unas reglas de prioridades con absoluta rigurosidad. Si existen diferentes paréntesis (se llama en este caso anidados), éstos se evalúan en primer lugar.

Tabla 2.8. Prioridad de operadores.

Tipo de operador	*Prioridad*	*Operador*	*Significado*
Paréntesis	1	( )	Máxima prioridad
Operadores aritméticos	2	^↑**	Exponenciación
	3	–	Cambio de signo
	4	*	Multiplicación
	4	/	División
	5	\	División entera
	6	MOD	Módulo
	7	+	Suma
	7	–	Resta
Operadores de relación	8	=	Igual que
	8	< >	Distinto
	8	<	Menor que
	8	>	Mayor que
	8	< =	Menor o igual
	8	> =	Mayor o igual
Operadores lógicos	9	NOT	Complemento lógico
	10	AND	AND (Y) lógico
	11	O	OR (O) lógico
	12	XOR	OR exclusiva
		EQV	Equivalencia
	13	IMP	Implicación

NOTA: *En algunas computadoras los signos + y – tienen prioridad 7 (posterior a la suma y resta).*

Las operaciones en el mismo nivel de la tabla 2.8 se realizan de izquierda a derecha. Para cambiar el orden de ejecución de las operaciones, se han de utilizar paréntesis. Las operaciones entre paréntesis se realizan siempre primero; dentro de los paréntesis se mantiene el orden normal de las operaciones.

5.2. Tratamiento alfanumérico

Como ya se ha comentado, una computadora no solamente trata con números sino que también es necesario el tratamiento de palabras. En la actualidad se manipulan «secuencias de letras», «frases», etc., y el «tratamiento» o «proceso de textos» se ha hecho imprescindible en el Software estándar de cualquier microcomputadora que se precie.

Como ya se definió en el párrafo 3.2.2., una variable «alfanumérica» o de «cadena» es una serie de caracteres, letras o números presentes en el teclado del ordenador. Para representar una variable alfanumérica se utilizan las mismas reglas que a las variables numéricas añadiéndole el símbolo ($). Variables alfanuméricas válidas son:

```
A$, A1$, B5$, VIC$
```

Las cadenas en su representación alfanúmerica deben estar encerradas entre comillas. *Casi todos* los caracteres del teclado pueden formar parte de una notación de cadena, con la excepción de las comillas ("), debido a que ellas ya forman parte de la cadena, indicando dónde comienza y dónde termina.

Ejemplos:

```
A$="MARIA"
B$="HOLA"
C$="1234"
D$="14 DE NOVIEMBRE DE 1982"
```

Una *cadena nula o vacía* es aquella que no contiene caracteres ni espacios en blanco (se llama también de longitud cero).

```
A$=""
```

Los espacios en blanco en las cadenas se consideran como caracteres que son tratados de igual forma que los restantes símbolos y caracteres.

La longitud de las variables alfanuméricas o de cadena es el número de caracteres que contiene. Suele ser de 255 ó 256 caracteres, aunque hay muchas computadoras que permiten tratar cadenas de longitud mayor. Las cadenas pueden ser de longitud cero.

Las cadenas se pueden definir mediante las sentencias LET, IMPUT, READ, DATA y PRINT.

El tratamiento de cadenas facilita la *interacción* con la computadora al permitir al usuario un diálogo alfanumérico en «modo directo» entre ambos.

CONCATENACION DE CADENAS

La concatenación de cadenas es la operación de suma de cadenas para formar una cadena de mayor longitud.

La suma se realiza mediante el signo «+».

```
"BUENOS DIAS"+"TRISTEZA"=BUENO DIASTRISTEZA
"HOLA"+"MORGAN"+"QUE TAL ESTAS"=HOLAMORGANQUE TAL ESTAS
```

Mediante la sentencia de asignación se pueden concatener variables de cadena:

```
A$="VALENCIA"
B$="BARCELONA"
A$+B$="VALENCIABARCELONA"
```

Las variables de cadena se pueden reasignar. Así las variables anteriores se pueden cambiar por:

```
A$=A$+"ZARAGOZA"
B$="JAEN"+A$+B$
```

La concatenación se expresa en los diferentes BASIC además con el símbolo (+), con (;), @ , AND, &, etc.

Ejemplo 1:

```
A$="PEPE"
B$="LUIS"
C$="GARCIA"
A$+B$+C$="PEPELUISGARCIA"
PRINT A$+B$+C$
PEPELUISGARCIA
```

Ejemplo 2:

```
A$="PET":B$=" ":C$="CBM"
PRINT A$+B$+C$
PET CBM
```

Mediante este procedimiento se ha dejado un espacio en blanco al sumar las dos cadenas A$ y C$, ya que se ha asignado a la variable B$ el espacio en blanco.

El mismo resultado se podría haber obtenido así:

```
PRINT A$+" "+C$
```

COMPARACION DE CADENAS

Las cadenas se pueden comparar mediante los operadores relacionales de igual forma que los valores numéricos.

Para ello es necesario conocer el código ASCII (vea el Anexo I) que representa los diferentes caracteres de la computadora. Es decir, cada letra, número o símbolo especial está representado por un número del 0 al 255, de modo que este criterio se seguirá para la comparación relativa. Normalmente las letras están comprendidas entre el código 65 y 90 precisamente en el mismo orden que el alfabético; 65 a la letra A, 66 a la letra B, etc. Los dígitos 0 a 9 entre el código 48 y 57, etc.

Sin embargo, excepto para los casos de símbolos especiales, no será preciso «normalmente» conocer todo el código ASCII ya que por lo general las letras del alfabeto siguen un orden creciente, así como los dígitos. El orden de comparación sería A<B, B<C, C<D, etcétera, y los números tendrían el siguiente orden: 0<1, 1<2, 2<3, etc.

Cuando se comparan dos cadenas, se comienza la comparación letra a letra comenzando por la izquierda hasta llegar al último carácter.

Ejemplo:

```
A$<B$
```

equivale a decir que la cadena A$ está clasificada alfabéticamente antes que la cadena B$.

```
A$="A"
B$="ABC"
```

Ello implica que A$<B$.

NOTAS

1. Todos los caracteres de una cadena se comparan secuencialmente, siendo el primer carácter distinto quien define la relación.
2. No es posible mezclar tipos distintos de variables, o sea, numéricas o de cadena. Más adelante se verá, al estudiar las funciones de cadena, el método para tratar ambas juntas.
3. La norma general de comparación es averiguar el código decimal equivalente en código ASCII del símbolo «*» es 142 y el código ASCII del símbolo $ es 36. Por tanto, si A$="*" y B$="$", resultará B$<A$.

6. PROGRAMACION DE LAS TECLAS DE FUNCION

Cuando se pulsa una tecla de función en BASIC en modo directo, una cadena de caracteres se envía a la computadora (caso de disponer de ellas en el teclado). Se pueden también utilizar las teclas de función como respuesta a sentencias INPUT o LINE INPUT.

Al encender la computadora, BASIC se activa con las teclas de función predefinidas descritas en el Capítulo 1. Si desea redefinir las cadenas asignadas a las teclas de función, lo puede conseguir con la sentencia KEY. El formato de esta sentencia es:

KEY n, X$

n es el número de la tecla de función, en el rango 1 a 10.

X$ es una expresión de caracteres que será asignada a la tecla; la longitud máxima de la cadena es 15 caracteres.

Como ejemplo introduzca

```
KEY 3,"De acuerdo"
```

Si ahora pulsa la tecla F3, visualizará en pantalla

```
De acuerdo
```

Para listar las cadenas predefinidas o asignadas a las teclas de función en la visualización, se utiliza la sentencia KEY LIST con el siguiente formato

KEY LIST

Esta sentencia/orden listará las cadenas asignadas a las 10 teclas de función.

Recuerde que las sentencias/órdenes KEY ON y KEY OFF mostrarán o borrarán la visualización de las teclas de función en la línea 25 del monitor.

`KEY ON`	*en pantalla de 40 caracteres se visualizan las cadenas asignadas a las cinco primeras teclas*
	en pantalla de 80 caracteres, se visualizan las 10 teclas de función
`KEY OFF`	*hace desaparecer las teclas de función de la línea inferior de la pantalla*

7. UTILIZACION DE LA IMPRESORA (LLIST/LPRINT)

Si posee una impresora deberá utilizarla con frecuencia no sólo para obtener resultados parciales o definitivos, sino para listados de programas y naturalmente sus «puestas a punto» y «depuración». La impresora de líneas debe ser una gran ayuda para escribir programas largos. Veamos los medios más frecuentes para su utilización.

VOLCADOS DE PANTALLA EN COMPUTADORAS IBM PC Y COMPATIBLES

Shift/PrtSc

Pulsando las teclas [⇧] (Shift) y [PrtSc], se escribe en la impresora todo cuanto se está visualizando en ese momento en la pantalla. A partir del momento de la pulsación de [⇧] y [PrtSc] cada línea de texto mostrada en su pantalla se imprimirá en la impresora.

Ctrl/PrtSc

Si posee el sistema operativo DOS 2.0 se puede obtener en la impresora una copia exacta de la visualización de la pantalla pulsando simultáneamente las teclas [Ctrl] y [PrtSc].

IMPRESION DE LISTADOS

La orden usual para obtener listados en la impresora es: LLIST.

Formato:

LLIST *línea* 1-*línea* 2

Actúa de igual forma que la orden LIST.

`LLIST 60-` *imprime todas las líneas hasta el final del programa, partiendo de la línea 60.*

IMPRESION DE RESULTADOS EN IMPRESORA

Cuando se deseen imprimir datos de un programa en la impresora, puede conseguirse con las sentencias LPRINT y LPRINT USING. Ambas trabajan igual que PRINT y PRINT USING en pantalla (la sentencia PRINT USING se estudiará en el capítulo 7).

Formato:

LPRINT *lista de expresiones o variables*

Si un punto y coma o espacios separan los elementos en la lista de expresiones o variables, cada elemento se imprime con un espacio después de él y para números positivos le precede un espacio adicional. Si los elementos están separados con comas, se imprimen al comienzo de las sucesivas sumas de impresión. Para más información véanse los capítulos 1 y 2.

```
10 X=50
20 LPRINT 30.458;69;3;X
30 LPRINT
40 LPRINT
50 LPRINT 30.458,69,3,X
```

Producirá la salida siguiente tras su ejecución.

```
30.458  69  3  50


30.458          69              3               50
```

NOTA: *En algunas impresoras LPRINT no tiene efecto y se sustituye por PRINT#n, siendo* n *el canal de salida a impresora (PRINT#*8 en Amstrad).

CAPITULO 3

Sentencias básicas iniciales

O. INTRODUCCION

En esta lección se le introducirá en las sentencias básicas de programación, de modo que a su terminación usted sabrá realizar programas elementales y podrá iniciar sus propios programas.

A lo largo de la lección se verán los siguientes conceptos:

— Sentencia, instrucción y línea de programa.
— Sentencia REM.
— Sentencia LET.
— Sentencia PRINT.
— Sentencia INPUT.
— Sentencia END.

1. ELEMENTOS DE UN PROGRAMA

Las tres partes constitutivas de un programa son las *sentencias, instrucciones* y *líneas.*

SENTENCIA

Una sentencia en BASIC está constituida por una palabra escrita en inglés, formada mediante caracteres normales, que instruye a la computadora para que realice una función determinada.

Sentencias válidas son READ, INPUT, END, etc.

INSTRUCCION

La instrucción en BASIC es la acción que se debe realizar dentro del programa cuando se realiza la sentencia.

Así como en lenguaje máquina y en lenguaje ensamblador la instrucción suele estar bien definida, no ocurre lo mismo en el lenguaje BASIC, en donde se confunden *instrucción* y *sentencia,* por lo que generalmente se suele emplear una palabra por la otra. Adoptaremos el convenio de referirnos a las sentencias que constituyen el programa, reservando la denominación «instrucciones» para casos muy concretos.

LINEA

La línea de un programa está constituida por:

1. Número de línea (por comodidad se suelen numerar de 10 en 10, 100 en 100, etc., aunque eso no significa que no se pueda escribir la línea 15, 22, etc).
2. Sentencia BASIC.
3. Algunas versiones de BASIC (COMMODORE, HP, APPLE, ZX Spectrum, etc.) admiten varias sentencias en una línea con tal de que sean separadas mediante un carácter separador (: en APPLE, COMMODORE; @ en HP).

Un programa en BASIC de complejidad media puede tener decenas, centenas e incluso millares de líneas.

2. COMENTARIOS EN EL PROGRAMA: SENTENCIA REM

Comenzamos la explicación de sentencias BASIC con aquella sentencia que permite la documentación de programas, aunque como luego se comentará, la computadora la ignora y no lo ejecuta.

La documentación de un programa incluye los organigramas, la descripción del programa y el listado del mismo. Se debe tener presente que los programas, no siempre (al contrario), son ejecutados por las mismas personas que lo hicieron, y aunque así fuese, al cabo de un cierto tiempo de realización, es fácil su olvido y en caso de una modificación o variación ésta resultará difícil si no se tuvo la precaución de documentar el programa debidamente. Por ello, la documentación de un programa es un concepto importante que todo programador siempre ha de tener presente.

La sentencia REM permite la inclusión de comentarios de tipo recordatorio o aclaratorio dentro de un programa. REM es la abreviatura de REMARK, o sea, comentarios, notas, etc.

Mediante la sentencia REM se pueden añadir al programa informaciones para documentar sus funciones, pero sin que tengan efecto en el mismo.

La misión principal de la sentencia REM es de ayuda al programador o a la persona que utilice el programa, facilitándole una información la más clara y exacta sobre los distintos pasos del mismo.

Esta sentencia REM no es ejecutada por la computadora y se llaman instrucciones «no ejecutables», pudiéndose situar en cualquier parte del programa. Sin embargo, la computadora las detecta y se presentará en el listado de sentencias al pulsar LIST y precisamente donde le corresponda por su numeración.

La labor de documentación es fundamental en un programa y suele ser interna y externa. La documentación externa son los organigramas, manuales de manejo del programa y todas aquellas notas que facilitan la puesta en marcha del mismo. La documentación interna es la que va incorporada al programa en forma de sentencias REM.

Esta sentencia suele ser rechazada por muchos programadores y es un gran error. Se debe pensar que con gran frecuencia los programas se manejan por personas distintas a quienes los elaboran, e incluso el paso del tiempo hace que los mismos programadores olviden su desarrollo. En estos casos el estudio del listado será tedioso e incluso a veces imposible de descifrar.

Formato: *n* REM *Palabras o mensaje*

n Número de línea.

Sentencias REM son las siguientes:

```
10 REM PROGRAMA DEMO
20 REM AUTOR LUIS GARCIA
30 REM SUBRUTINA PRINCIPAL
```

Las sentencias REM se incluyen al principio del programa para incluir el nombre del programa, nombre del autor y otros datos de interés. En el resto del programa se emplean intercaladas a fin de explicar lo que se ejecuta realmente.

```
10 REM PROGRAMA DEMO 1
20 REM AUTOR MORTIMER
30 REM PROGRAMA DEMOSTRATIVO
40 REM DE OBTENCION DE GRAFICOS
..........
..........
```

Es corriente utilizar líneas en blanco antes y después de una sentencia REM. El objetivo es simplemente resaltar los comentarios del resto del programa. Con ello se aumenta la claridad en la documentación.

```
10 REM DEMO
20 REM
30 REM
40 REM *** PROGRAMA DEMO ***
50 REM
60 REM
```

En ocasiones, es conveniente incluir un comentario explicativo de sentencias claves mediante REM; sin embargo, es mucho más eficaz colocar el comentario en la misma línea. Algunas versiones de BASIC admiten separadores de comentarios como (') o (!). (Las computadoras IBM PC, MSX, Amstrad, etc. adoptan (') y HP adopta (!)).

```
10 GOSUB 1000          !SUBRUTINA LECTURA DE NOMBRES
200 GOSUB 1500         !IMPRESION DATOS EN IMPRESORA
```

La línea 10 llama a la subrutina que comienza en la línea 1000 y que realiza la lectura de nombres. La línea 200 llama a la subrutina que comienza en la línea 1500 y que imprime datos en impresora. (Sentencia GOSUB «llamada a subrutina».)

El otro medio para escribir la sentencia REM es utilizar el apóstrofo o símbolo simple de comillas (') en lugar de REM que significa la misma cosa. Por ejemplo:

```
10 GOSUB 1000: REM SUBRUTINA lectura de nombres
10 GOSUB 1000: 'Subrutina de lectura de nombres
```

son equivalentes. (La sentencia GOSUB es «llamada a subrutina»; en el capítulo 9 se estudiará con detenimiento.) Por ahora, basta con saber que el programa bifurca a la línea 1000.

Buena práctica de programación:

El mejor medio para asegurar que sus programas sean claros y fáciles de comprender es establecer algunas reglas de buena práctica de programación. Algunas de estas reglas para recordar y aplicar en sus programas se dan a continuación. En primer lugar, todos los programas deben tener:

1. Un título.
2. Una fecha.
3. El nombre del autor.
4. Comentarios explicativos, en síntesis, del objetivo del programa.

En segundo lugar, debe separar las secciones por comentarios de ellas o definir cada uno de los siguientes elementos en su programa:

1. Referencias (fuentes de información).
2. Variables (nombres de los elementos modificables).
3. Constantes (nombres y valores de los elementos permanentes).
4. Entradas (fuentes de información del programa).
5. Salidas (resultados de las operaciones).
6. Programa principal (donde se realizan las operaciones).
7. Subrutinas (operaciones repetidas que se realizan fuera del programa principal).
8. Datos (información a procesar).

Aunque algunos de los conceptos no le son familiares, en este momento de su estudio del BASIC, no se preocupe por ellos, se explicarán en capítulos posteriores. Lo importante, ahora, es que vea la necesidad de identificar cada elemento en su programa.

```
10 REM **************
20 REM *    AUTOR   *
30 REM * J.R.Mackoy *
40 REM **************
50 REM
60 REM Impresion en pantalla de los caracteres ASCII
70 REM y sus valores decimales equivalentes
80 REM
90 CLS           'Limpiar pantalla
100 REM Comienzo programa principal
110 REM
120 FOR N=0 TO 255
130 PRINT CHR$(N),N
140 NEXT N
150 REM
160 REM -Fin del programa principal-
170 REM
180 PRINT CHR$(7) 'Suena un pitido cuando se termina
999 END
```

En tercer lugar, utilizar abundancia de espacios y líneas en blanco. Ello es especialmente importante para programas largos, que de no ser así, se pueden hacer ilegibles. Se ha de evitar poner muchas sentencias en una sola línea con el separador dos puntos (:), a menos que estas sentencias sean muy cortas y claras. El programa anterior podría quedar así:

```
10 ' **************
20 ' *    AUTOR   *
30 ' * J.R.Mackoy *
40 ' **************
50 '
60 ' Impresion en pantalla de los caracteres ASCII
70 ' y sus valores decimales equivalentes
80 '
90 CLS           'Limpiar pantalla
100 ' Comienzo programa principal
110 '
120 FOR N=0 TO 255
130 PRINT CHR$(N),N
140 NEXT N
150 '
160 ' -Fin del programa principal-
170 '
180 PRINT CHR$(7) 'Suena un pitido cuando se termina
999 END
```

Inconveniente:

Aunque la documentación mediante sentencias comentario es muy necesaria, tampoco se debe abusar de ellas, dado que si bien la computadora no las ejecuta, sí ocupan espacio físico de memoria. Por ello deberá tener cuidado, sobre todo en el caso de programas largos, para evitar que las sentencias REM (o sus equivalentes) le ocupen espacio de memoria excesivo y tenga que proceder posteriormente a borrarlas, caso de que se agote la memoria libre para el usuario. Esta preocupación será mayor si su microcomputadora tiene menos de 16 K, cosa que, por suerte, ya prácticamente no sucede en ninguna computadora —incluso doméstica o familiar— como no sea un modelo antiguo.

3. SENTENCIAS DE ASIGNACION: LET

La sentencia LET se conoce como asignación y permite de un modo flexible asignar cualquier número, expresión algebraica o expresión de cadena a variables de cualquier tipo.

El *formato* de la sentencia LET es similar a la igualdad matemática:

n LET *variable = valor o expresión*

n	Número de línea.
variable	Cualquier variable numérica o de cadena válida.
valor o expresión	Cualquier valor numérico o expresión válida.

Una de las aplicaciones más usuales de la sentencia LET se presenta cuando un valor se ha de utilizar varias veces.

Ejemplo:

```
10 LET X=3.45
20 LET X=425
30 LET C1=B2
40 LET N$="Nombre"
50 LET A$="MORGAN Y MORTIMER"
60 LET A$=N$
```

La asignación se obtiene situando el valor o la expresión a la derecha del sino igual, y el nombre de la variable, elegida arbitrariamente, a la izquierda del signo. En el caso de que la asignación se realice mediante otra variable o expresión, el valor que se asigna es el que tenga en ese momento el segundo término.

```
10 LET N=4.25
20 LET A=N
30 LET A$="MARIA"
40 LET B$=A$
```

En el programa anterior, la variable N toma el valor 4.25. La variable A toma el valor N o, lo que es igual, 4.25. Las variables A$ y B$ toman el valor de MARIA.

En el caso de asignación de expresiones numéricas el resultado de las mismas realiza la misma función.

```
10 LET A = 3*5-4
```

La variable A toma el valor 11, resultado de 3*5−4.

Es necesario destacar que si bien la sentencia LET es muy similar a una ecuación algebraica, existen expresiones válidas en LET que no tienen sentido algebraicamente.

```
10 LET N = N+1
```

La línea 5 significa que se ha incrementado el valor de la variable N en 1 y que se ha asignado este valor a la misma, reemplazando el valor que tenía anteriormente.

Ejemplo:

```
10 LET N=2
20 LET N=N+4
30 LET N=N^2
40 LET B=5
50 LET N=N+1
60 LET C=9
70 LET A=B+N
```

Si se ejecuta este programa mediante la orden RUN, los valores que se asignan a cada variable al llegar a su línea respectiva serían los siguientes:

Línea 10	N=2
Línea 20	N=6
Línea 30	$N=(6^2)=36$
Línea 40	B=5
Línea 50	N=36+1=37
Línea 60	C=9
Línea 70	A=5+37=42

NOTAS:

1. *Si en un programa aparece una variable a la que no se ha asignado previamente un valor, se debe entender que su valor es cero.* Sin embargo, algunas computadoras exigen la inicialización a cero u otro valor determinado de todas las variables, ya que en caso contrario producen error.

2. *Algunas versiones de BASIC permiten mayor flexibilidad que otras en la escritura de sentencias LET.*

— Existen versiones de BASIC que permiten asignar el mismo valor a varias variables en una única sentencia LET.
— Las versiones modernas de BASIC permiten la omisión de la palabra LET.

```
10 A=6
20 B=5
30 A$="MARIA"
40 B$="PEPE LUIS"
50 A=B*2-4+5
```

Ejemplo:

Realizar un programa que encuentre el valor decimal de la expresión: $\frac{2/5+3/7}{3/4-1/3}$

```
10 LET N=2/5+3/7
20 LET D=3/4-1/3
30 PRINT N/D
40 END
```

Al ejecutar el programa:

```
RUN
 1.9885714
```

- Otras dos versiones del programa pueden ser:

```
10 LET F=(2/5+3/7)/(3/4-1/3)
20 PRINT F
30 END

RUN
 1.9885714

10 PRINT (2/5+3/7)/(3/4-1/3)

RUN
 1.9885714
```

Regla práctica: En lo sucesivo, normalmente se suprimirá la palabra LET de la sentencia ASIGNACION. *Ejemplo:* A=6, B=B+1, etc. (Sin embargo, las populares computadoras familiares de SINCLAIR, ZX Spectrum exigen siempre la palabra LET.)

4. ENTRADA DE DATOS: INPUT

La sentencia INPUT permite introducir datos numéricos o alfanuméricos a la computadora durante la ejecución del programa.

Mediante la sentencia INPUT se permite la intercomunicación directa entre la computadora y el usuario a través del teclado o del terminal.

Formato:

n INPUT $variable_1$, $variable_2$, ...

n Número de línea.
$variable_i$ Nombre de la variable válida (i=1, 2, ...).

Ejemplos:

```
10 INPUT A, B, C
20 INPUT M
30 INPUT A$, B$
40 INPUT A(I), B(I)
```

Las variables A(I), B(I) se llaman de índice o subíndice y se estudiarán en un capítulo posterior.

Al *ejecutarse* el programa, cada vez que se encuentre una sentencia INPUT se interrumpirá el proceso y aparecerá un signo de interrogación (?) en la pantalla que le hace saber al usuario u operador que la computadora está esperando datos.

El signo de interrogación se escribe normalmente al principio de la línea y la ejecución del programa se interrumpe hasta que se introducen los datos solicitados. El operador deberá introducir los datos y pulsar la tecla RETURN.

NORMAS PARA LA INTRODUCCION DE DATOS

1. Los datos deben corresponder en tipo y número con la lista de variables de la sentencia INPUT correspondiente. Es decir, se deben introducir números para variables numéricas y caracteres alfanuméricos para variables de cadena. Si no lo hace así, la computadora le dará un mensaje de error del tipo: ?REDO FROM START, TYPE MISMATCH ERROR, etc.

2. Si se introducen menos datos de los solicitados, en pantalla aparecerá otro signo de interrogación que significa «FALTA DE DATOS» y habrá de introducir los datos que falten.
3. Si se introducen más datos de los necesarios, se ignorarán los sobrantes por la máquina y aparecerá un mensaje de error «EXTRA IGNORED».
4. Los datos a introducir se deben separar por comas.
5. Si al introducir una cadena se empieza con espacios en blanco, éstos se pierden. Si se escribe una coma, se perderá todo lo que venga detrás (o se considerará un nuevo dato si la computadora está esperando recibir varios).
6. El inconveniente anterior se puede obviar encerrando la cadena entre comillas.

Ejemplo:

```
10 INPUT A, B, C$
```

Al ejecutarse esta línea aparecerá:

```
?
```

que está pidiendo la introducción de datos adecuados y separados por coma:

```
?45,221,LUIS
```

Una vez escritos los datos se pulsará RETURN.

La sentencia INPUT admite una variante en algunas versiones de BASIC que permite la introducción de un mensaje dentro de la misma sentencia, con lo que facilita la intercomunicación usuario-computadora, ya que de no disponer de esta posibilidad se deberá recurrir normalmente a la sentencia PRINT.

Formato: *n* INPUT *"MENSAJE"; variable*$_1$, *variable*$_2$, ...

n	Número de línea.
variable	Variable válida.
MENSAJE	Frase o mensaje que suele indicar el tipo de datos que se ha de introducir.
; (separador)	En algunas versiones BASIC se sustituye por (,).

Ejemplo 1: Supongamos que se necesita introducir el nombre y apellidos de un alumno, la edad y la fecha de nacimiento. La forma más adecuada de programarlo sería:

```
10 INPUT "NOMBRE Y APELLIDOS";N$
20 INPUT "EDAD";E
30 INPUT "FECHA DE NACIMIENTO";F$
```

En este caso el programa al ejecutarse presentaría los siguientes resultados:

```
NOMBRE Y APELLIDOS?
EDAD?
FECHA DE NACIMIENTO?
```

Se introducirán tres respuestas consecutivas: LUIS GARCIA GARCIA, 19, 4 DE OCTUBRE 1967.

Ejemplo 2: El programa 1 se podría hacer de esta otra forma:

```
10 INPUT "NOMBRE Y APELLIDOS, EDAD, FECHA DE NACIMIENTO"; N$, E, F$

RUN
NOMBRE Y APELLIDOS, EDAD, FECHA DE NACIMIENTO?
LUIS GARCIA GARCIA, 19, 4 DE OCTUBRE DE 1967
```

Se introducen por teclado

Ejemplo 3: La sentencia siguiente permite la introducción de la fecha del día en un programa.

```
10 INPUT "FECHA DEL DIA: DD, MM, AA": A, B, C
```

Este tipo de mensaje es muy usual en programación. Al ejecutarse esta sentencia aparecerá en pantalla:

```
FECHA DEL DIA: DD, MM, AA? 15, 4, 82
```

El operador ha de introducir el día, mes y año, todos ellos con dos cifras decimales, separados por comas.

Ejemplo 4: En caso de no poder utilizar el formato de INPUT con mensaje, tal como se muestra en los ejemplos 1, 2 y 3, las sentencias anteriores se podrían escribir así:

```
10 PRINT "NOMBRE Y APELLIDOS"              Ejemplos 1 y 2
20 INPUT N$
30 PRINT "EDAD"
40 INPUT E
50 PRINT "FECHA DE NACIMIENTO"
60 INPUT F$

10 PRINT "FECHA DEL DIA: DD, MM, AA"       Ejemplo 3
20 INPUT A, B, C
```

5. LINE INPUT (LINPUT)

La sentencia INPUT tiene la limitación del empleo de espacios en blanco y signos coma (,), al introducir datos tipo cadena y en ocasiones limitación en el número de caracteres a introducir. Si trata de introducir por teclado, en respuesta a INPUT, una cadena de caracteres con comas incluidas, se obtendrá un mensaje de error «Redo from star» (vuelva a comenzar) o similar.

Mediante la sentencia LINE INPUT se evita el inconveniente anterior. La sintaxis y comportamiento de LINE INPUT es idéntica a INPUT excepto en las siguientes diferencias:

1. LINE INPUT solamente acepta cadenas de caracteres. Todos los caracteres introducidos en respuesta al «mensaje», incluyendo comas y comillas se asignarán a la variable de cadena.
2. Solamente se puede introducir una cadena de caracteres con cada sentencia LINE INPUT.
3. LINE INPUT no devuelve un signo de interrogación, como lo hace INPUT, cuando se ejecuta.

Los datos introducidos por teclado pueden tener hasta 254 o 255 caracteres de longitud, según computadora.

Formato:

n LINE INPUT *"mensaje"; variable de cadena*

El mensaje, al igual que INPUT, puede ser cualquiera que vaya entre comillas. Es opcional y si se incluye deberá separarse de la variable de cadena por un punto y coma:

```
10 LINE INPUT A$
20 LINE INPUT "NOMBRE": N$
```

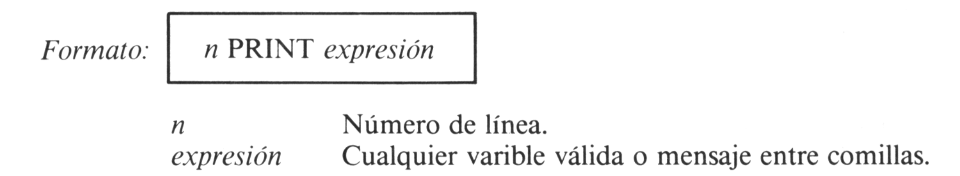
NOTA 1: La versión CBASIC admite la sentencia INPUT LINE que es similar a LINE INPUT. Otra variante suele ser LINPUT, que adoptan algunas computadoras.

NOTA 2: Las sentencias INPUT y LINE INPUT son muy útiles en programación para conseguir la intercomunicación operador/computadora. Sin embargo, si los datos a introducir son muchos, la entrada de información es lenta y a veces tediosa.

Es preciso tener en cuenta que los datos introducidos mediante INPUT no se pueden almacenar para su uso posterior en otras ocasiones de empleo del mismo programa, por lo que si se desean conservar será preciso recurrir a otra sentencia (READ/DATA) o al tratamiento de ficheros mediante un dispositivo externo de memoria.

6. PRESENTACION/IMPRESION DE DATOS: PRINT

La sentencia PRINT permite la impresión de datos de salida de cualquier tipo, en pantalla o en periférico, como puede ser la impresora.



Formato: *n* PRINT *expresión*

n Número de línea.
expresión Cualquier varible válida o mensaje entre comillas.

La sentencia PRINT se puede utilizar de diferentes formas:

— Impresión de resultados de cálculos definidos en el interior de la sentencia.

Ejemplo:

```
10 PRINT 4*6/1.5
20 PRINT 2.5+8.9,7.5*8.4
30 PRINT 4*5
40 END
```

RUN

```
16
11.4            63
20
```

— Impresión de valores calculados anteriormente:

```
10 A=5
20 B=4
30 C=A/B
40 PRINT A, B, C
50 END

 5              4              1.25
```

El programa imprime cada valor A, B y C en una zona.

— Impresión de mensajes o comentarios.

```
10 PRINT "ESTE PROGRAMA CALCULA"
20 PRINT "EL CUADRADO DE 6"
30 PRINT "ASI COMO EL CUBO DE 4"
40 PRINT 6^2, 4^3
50 PRINT "FINAL DEL TRABAJO"
60 END
```

```
RUN
ESTE PROGRAMA CALCULA
EL CUADRADO DE 6
ASI COMO EL CUBO DE 4
 36            64
FINAL DEL TRABAJO
```

— Impresión de líneas en blanco.

```
10 PRINT "ESTE PROGRAMA ESCRIBE"
20 PRINT "LINEAS EN BLANCO"
30 PRINT
40 PRINT
50 PRINT "SE HAN DEJADO 2 LINEAS EN BLANCO"
60 PRINT "FIN"
70 END

RUN
ESTE PROGRAMA ESCRIBE
LINEAS EN BLANCO


SE HAN DEJADO 2 LINEAS EN BLANCO
FIN
```

— Impresión de mensajes y variables numéricas o de cadena en la misma línea.

```
10 A=3000
20 B=45
30 C=720
40 PRINT "EL VALOR DE A ES IGUAL A"; A
50 PRINT "EL VALOR DE B+C ES"; B+C
60 END

RUN
EL VALOR DE A ES IGUAL A 3000
EL VALOR DE B+C ES 765
```

6.1. Normas de tratamiento sentencia PRINT

1. La sentencia PRINT, sola, sin varibles o cadenas, imprime o visualiza una línea en blanco que se suele utiliar en la práctica para el espaciado vertical y dar mayor claridad a los datos de salida.
2. Los números positivos están siempre precedidos de un espacio en blanco.
3. Los números negativos son precedidos por un signo menos (−).
4. La impresión de las cifras de un número dependen de la versión BASIC. Si se desea conocer con exactitud, deberá consultar el *Manual de Instrucciones* de su computadora.
5. Las cadenas se deben encerrar entre comillas.
6. La preparación del formato de salida de datos (*formateado* en la jerga informática) en pantalla o en impresora se realiza mediante los signos de puntuación (,) y (;). El empleo de comas y puntos y comas permitirá que la presentación de resultados se ajuste exactamente al modelo de presentación deseados. Dado su importancia, el tratamiento de formatos se verá en el siguiente párrafo más detenidamente.

6.2. Formatos de pantalla

Las pantallas utilizadas en los sistemas de computadora dependen del generador de caracteres de los mismos; sin embargo, el formato estándar suele ser de 24 ó 25 filas y 40 u 80 columnas. En la actualidad, en microcomputadoras se tiende hacia 25 × 80 filas × columnas.

Formatos no estándar son:

ZX Spectrum	24 ×	32 *filas × columnas*
DRAGON-64	16 ×	32 *filas × columnas*
SINCLAIR QL	25 ×	85 *filas × columnas*
AMSTRAD	25 ×	20 *filas × columnas (en modo 0)*

Nuestro estudio lo realizaremos para las pantallas estándar que se consideran más utilizadas en la actualidad: 24 × 40 y 25 × 80 (filas × columnas). Entre otras computadoras, son utilizadas por APPLE II, Commodore, TRS-80, IBM PC, MSX, Amstrad, etc.

Así pues, una pantalla de 25 × 40 permitirá la representación máxima de 1000 caracteres.

A efectos del tratamiento del símbolo (,) en la sentencia PRINT, la pantalla se divide en *zonas* igualmente espaciadas y que suelen ser 10, 14 o 20 espacios para presentaciones de 40 u 80 columnas. (La computadora DRAGON-64 asimismo en otras dos zonas.) En nuestro caso consideramos cuatro zonas, cada una separada por 10 espacios o columnas.

Así, la primera columna se denominará *columna 1,* y las siguientes *columna 2, ..., ..., columna 40, ...* De igual modo el número de filas se denominará *fila 1, fila 2, ... fila 25.* Esta numeración comienza tanto para las filas como para las columnas, por la parte superior izquierda o posición inicial de pantalla.

Así pues, cada zona comienza en las columnas 1, 11, 21 y 31.

Precaución: Lea en el manual de su computadora cuál es la presentación de su pantalla y el número de zonas en que se divide a efectos de impresión.

6.3. Presentación de formatos mediante comas y punto y coma

El empleo de símbolos (,) y (;) permite que la presentación de resultados se ajuste exactamente al modelo de configuración deseado.

EMPLEO DEL PUNTO Y COMA (;)

El empleo de un punto y coma (;) en una sentencia PRINT permite la escritura de variables o cadenas alfanuméricas en una sola línea y con la mínima separación.

Introduzca y ejecute el siguiente programa:

```
10 PRINT "14+16="; 14+16

RUN
14+16= 30
```

Se observará que el punto y coma ha hecho que el resultado de la operación se escriba *inmediatamente* a continuación del mensaje «14+16», habiendo dejado dos espacios en blanco entre el signo igual y el dígito 3 de 30 (algunas computadoras no dejan ningún blanco, como es el caso de ZX Spectrum).

Introduzca y ejecute ahora el siguiente programa:

```
10 PRINT "TENEMOS"; 15*4; "LAPICES Y"; 4*3; "PLUMAS"

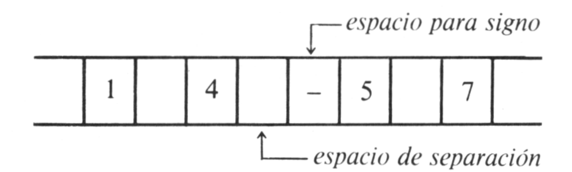
RUN
TENEMOS 60 LAPICES Y 12 PLUMAS
```

En resumen, el símbolo (;) actúa como un separador.

Reglas prácticas:

a) Los puntos y comas en las sentencias PRINT dejan un espacio en blanco detrás del número siempre, y un espacio en blanco delante para el signo en caso de cantidades o variables

numéricas; en caso de variables de cadena o frases encerradas entre comillas no se dejan espacios en blanco entre ellas.



Introduzca y ejecute el programa siguiente:

```
10 PRINT 1:4;-5;7
20 A=5 : B=7
30 A$="MARIA"
40 B$="GARCIA"
50 PRINT A; B
60 PRINT A$; B$

RUN
 1  4 -5  7
 5  7
MARIAGARCIA
```

b) Si al final de una línea con sentencia PRINT se coloca un punto y coma, al ejecutarse la siguiente sentencia PRINT no se salta línea, sino que los resultados se encadenan de acuerdo con la norma *a),* imprimiéndose a continuación de lo último escrito.

```
10 PRINT "RADIO DEL CIRCULO";
20 A=25
30 PRINT A
40 PRINT "FIN DEL ";
50 PRINT "MENSAJE"
60 END

RUN
RADIO DEL CIRCULO 25
FIN DEL MENSAJE
```

EMPLEO DE COMAS

Si suponemos la pantalla (otra distribución normalizada, suele ser 15, 29, 43,... para 80 columnas) de 40 columnas, se divide ésta en cuatro zonas que comienzan en las columnas 1, 11, 21 y 31. Mediante las comas, se separan las variables o expresiones de la sentencia PRINT, de modo que cada una comienza a escribirse en la siguiente zona.

```
10 A=5: B=6: C=7: D=9
20 PRINT A, B, C, D

RUN
 5                   6                   7                   9
```

Destaquemos que como la presentación de cantidades numéricas deja un espacio en blanco a su izquierda siempre, la impresión de resultados numéricos comienza siempre en las columnas 2, 12, 22 y 32 (o bien 16, 30, 44, ... en 80 columnas). En caso de ser variable de cadena o mensajes alfanuméricos comenzaría la impresión en las columnas 1, 11, 21 y 31 (o bien 15, 29, 43, ...).

Reglas prácticas:

a) Un separador coma hace que se escriba la variable o expresión alfanumérica que viene a continuación en la siguiente zona de la pantalla.

b) Si una sentencia PRINT se termina con una coma (,), la impresión resultante de ésta se encadenará con los resultados de la siguiente sentencia PRINT, comenzando esta última en la siguiente zona de la misma línea de pantalla, y en caso de haberse agotado las cuatro zonas de la línea comienza en la primera zona de la siguiente línea de pantalla.

```
10 PRINT A, 54, "MARIA",
20 PRINT "GONZALEZ"

RUN
 0                  54                  MARIA               GONZALEZ
```

Si un número o secuencia de caracteres «invade» el campo siguiente, el número o secuencia de caracteres siguientes se imprimirá en el primer campo libre, pudiendo pasarse incluso a la fila siguiente.

Recomendación: Tenga presente que una de las características más sobresalientes en el desarrollo de un programa es la presentación de resultados en pantalla o en impresora, por lo que será necesario que el análisis de los formatos de salida se realice con detenimiento antes de la ejecución del programa, o bien durante la fase de corrección o puesta a punto del mismo.

Las impresoras comerciales suelen tener 80 ó 132 caracteres normalmente, por lo que será necesario que el análisis de los formatos de salida se realice con detenimiento antes de la ejecución del programa, o bien durante la fase de corrección o puesta a punto del mismo.

Las impresoras comerciales suelen tener 80 ó 132 caracteres normalmente, por lo que será preciso recurrir a la escritura de (,) y (;) para aprovechar al máximo sus características, enlazando dos o más sentencias PRINT mediante los mencionados símbolos.

Posteriormente se verán las funciones de formatos, SPC, TAB y POS, que, junto con los símbolos (,) y (;), facilitarán considerablemente la presentación de resultados.

7. FIN DE PROGRAMAS: END

La sentencia END indica el FINAL de un programa. Ya se ha visto en varios programas de párrafos anteriores.

Al ejecutarse esta sentencia se instruye a la unidad central de la computadora para que dé por terminadas las tareas a realizar por el programa, deteniéndose a continuación en espera de nuevas órdenes.

Formato:



n Número de línea.

— Normalmente sólo existe un sentencia END en cada programa.
— El número de orden de la línea correspondiente debe ser el más alto del programa.

No es necesario la terminación de un programa mediante la sentencia END, y de hecho con frecuencia no se incluye (la computadora termina el programa con la última línea ejecutada sea o no END). Sin embargo, *hay computadoras* (HEWLETT PACKARD) *que dan error si no se incluye en el programa,* aunque este error evidentemente no tiene importancia.

Como buena norma de programación le aconsejamos termine todos sus programas con una sentencia END.

8. RESUMEN

La sentencia asignación LET permite asignar el valor de la variable o expresión situada a la derecha a la variable situada a la izquierda.

```
LET S=V
```
A la variable S se le asigna el valor de V.

En la mayoría de los BASIC actuales la sentencia asignación se representa sin la palabra clave LET, o a lo sumo es opcional. (*Recuerde* que SINCLAIR no sigue esta regla.)

```
A=5
B=3*A
```

El signo (=) no tiene el sentido de igualdad matemática, sino el de *asignación.*

En todas las instrucciones de asignación, el cálculo de la expresión situada a la derecha del signo (=) se realiza *antes* de la asignación final del resultado a la variable indicada a la izquierda del signo (=).

A la izquierda del signo (=) no debe figurar más que un único nombre de variable válida. La asignación se hace siempre de la derecha hacia la izquierda.

```
A1=3*5
B2=A+B-C
```

La sentencia INPUT permite la introducción de datos en la computadora mediante teclado durante la ejecución de un programa.

```
INPUT A, B, C
```

La ejecución de una sentencia INPUT interrumpe temporalmente el desarrollo del programa y advierte al usuario (generalmente mediante un signo de interrogación ?) que debe introducir datos.

El usuario introduce entonces los datos bajo la forma de constantes numéricas o cadenas de caracteres, terminando la operación pulsando como siempre la tecla RETURN.

En una línea de programas puede haber varias sentencias separadas por separadores, normalmente (:).

Si un programa lleva varias sentencias INPUT con la forma citada, el usuario se arriesga a equivocarse e introducir valores numéricos que no se correspondan con el orden de las sentencias e incluso equivocarse en el orden de las variables. Para evitar este inconveniente existen dos procedimientos:

a) Hacer preceder a cada sentencia INPUT de una sentencia PRINT que explique al usuario lo que debe hacer.

```
100 PRINT "NUMERO DE HORAS"
120 INPUT H
```

b) Las versiones modernas de BASIC admiten una forma más potente para la sentencia INPUT.

```
100 INPUT "NUMERO DE HORAS"; H
```

El mensaje entre comillas funciona igual que una sentencia PRINT.

La sentencia PRINT permite imprimir las cantidades o variables numéricas que vayan a continuación de dicha palabra clave, o bien las variables alfanuméricas y mensajes o palabras encerradas entre comillas.

Cada sentencia PRINT comienza la impresión en una nueva línea, salvo si la última sentencia PRINT se termina por una *coma* (,) o por un *punto y coma* (;).

La sentencia PRINT escrita sola provoca *un salto de línea.*

```
100 PRINT
200 PRINT
300 PRINT
```

Salto de tres líneas en pantalla

Todas *las cadenas de caracteres* que se deban imprimir como mensajes se deben escribir entre *comillas* en una sentencia PRINT. Una sentencia PRINT puede imprimir *un único mensaje* o *mensajes entre valores de las variables.*

Una coma, así como un punto y coma, pueden separar las variables (o los mensajes alfanuméricos) en una sentencia PRINT. *La coma* provoca la impresión a partir de *columnas fijas* en una línea. *El punto y coma* provoca *la impresión continuada* en una línea.

La coma define zonas en la longitud total de la línea. El número de columnas (o caracteres) en una zona varía de una computadora a otra, pero en general son 10 en pantalla de 40 y 15 o 20 en líneas de 80 caracteres.

Cada valor de variable se imprime en una zona independiente de la longitud del número a imprimir. Una longitud de caracteres puede ser impresa en varias zonas.

Las cadenas alfanuméricas se imprimen a partir de *la columna más a la izquierda de una zona.* Las variables numéricas *dejan un espacio para el signo,* sea este positivo o negativo.

La sentencia END es la última de un programa.

REGLAS DE USO DE LA SENTENCIA INPUT

— El número de constantes a introducir debe ser igual al número de variables numéricas de la sentencia INPUT. El número de cadenas de caracteres también será igual al número de variables de cadenas incluidas en la sentencia PRINT.

— Si el número de datos introducidos es mayor o menor que el número de datos solicitados, aparecerá un mensaje de error o ignorará los datos sobrantes.

```
10 INPUT A, B, C

RUN
?4, 5
??
```

<RETURN>
(Mensaje indicativo de falta de datos, deberá introducir un tercer dato.)

```
10 INPUT A,B

RUN
4, 5, 6, 7
```

(Aparece el mensaje EXTRA IGNORED, significa que los datos 6 y 7 los ignora, o bien ?REDO FROM START —vuelva a comenzar—.)

La sentencia REM (abreviatura de REMARK) permite documentar programas.

Un buen programa debe constar de las suficientes sentencias REM como para permitir la corrección o modificación del pograma con el paso del tiempo, bien por el mismo programador que lo confeccionó, bien por cualquier otra persona distinta.

El inconveniente de la sentencia REM es que ocupa espacio de memoria útil y, en consecuencia, deberá evitarlas si tiene problemas de limitación de memoria.

Los métodos usuales de documentación pueden ser éstos:

1)
```
100 REM CALCULO DE RESULTADOS
120 A=B+C
140 ...
200 REM PRESENTACION DE RESULTADOS
220 N=N+1
240 ...
... ...
```

2)
```
100 A=B+C: REM CALCULO DE RESULTADOS
... ...
... ...
200 N=N+1: REM PRESENTACION DE RESULTADOS
... ...
```

Determinadas versiones de BASIC admiten como separador, en el caso de la sentencia REM, los símbolos (!) o (′).

```
100 GOSUB 1000          !SUBRUTINA
... ...
200 GOSUB 2000          !SUBRUTINA
```

FORMATOS DE SENTENCIAS VALIDAS

```
10 REM
20 REM NOTAS DE ALUMNOS
30 INPUT N
40 INPUT A$
50 INPUT "INTODUZCA NOMBRES"; N$
60 PRINT A
100 PRINT A$, B$, C$
120 PRINT "MARIA LOPEZ"; "GARCIA"
140 LET A=A+1
160 LET A=7*4-5
180 LET A=B^2
200 A=N*2-3
220 B=B^2-4
240 N=N+4
260 N=N+1: A=A+1: C=C+1
280 REM NOTAS PARCIALES
300 PRINT A, B, C: REM IMPRESION DE DATOS
320 INPUT A, B, C
340 INPUT A$, B$, C$
360 INPUT "NOTAS DE CLASE"; N
400 END
420 PRINT A$, B$, C$, "IMPRESION DE RESULTADOS"
500 PRINT "MORTIMER";N       :REM EDAD DE MORTIMER
600 PRINT "MCKOY";M          !SUELDO DE MCKOY
700 PRINT "FLANAGAN"         ′MAFIOSO
```

La anterior lista de sentencias no constituye un programa específico, sólo se ilustra como ejemplos de sentencias válidas en lenguaje BASIC (de hecho los 2 separadores de sentencias REM (′ y ! no pueden existir al mismo tiempo).

PROGRAMA DE APLICACION

Resolución de una ecuación de segundo grado

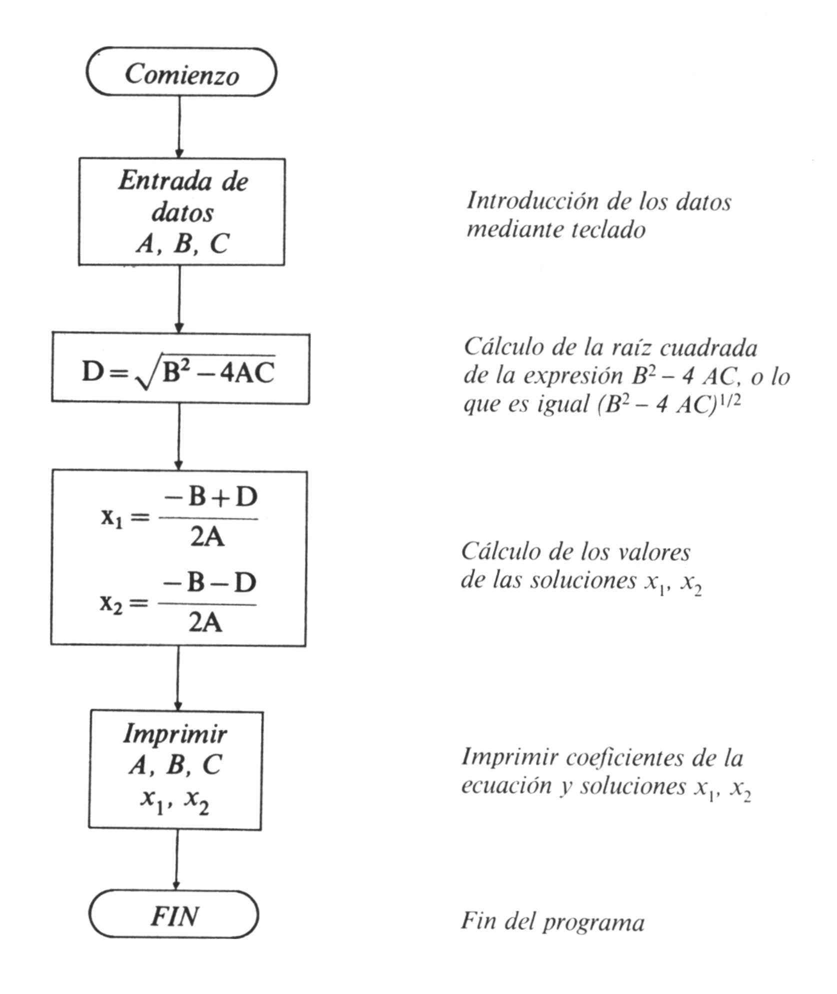
Confeccionar un programa que permita obtener las soluciones X1 y X2 de una ecuación de segundo grado conociendo los valores de los coeficientes que se introducirán mediante sentencias INPUT.

Solución:

Se desea resolver la ecuación siguiente: $Ax^2 + Bx + C = 0$.
Las soluciones de esta ecuación son:

$$x_1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \qquad x_2 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

El organigrama del programa sería el siguiente:



La codificación del programa sería ésta:

```
5 REM DEMO 1
10 REM CALCULO ECUACION SEGUNDO GRADO
20 REM ENTRADA DE DATOS
25 PRINT "INTRODUZCA COEFICIENTES"
30 INPUT A, B, C
40 D=(B^2-4*A*C)^.5
50 X1=(-B+D)/(2*A)
60 X2=(-B-D)/(2*A)
70 REM PRESENTACION DE LOS RESULTADOS
80 PRINT
90 PRINT
100 PRINT "EL VALOR DE LOS COEFICIENTES ES"
110 PRINT "A="; A, "B="; B, "C="; C
115 PRINT
120 PRINT "SOLUCIONES"
140 PRINT "X1="; X1
160 PRINT "X2="; X2
200 END
```

Al ejecutarse este programa mediante el comando RUN se presentarían los siguientes mensajes en pantalla con sus resultados correspondientes:

```
RUN
INTRODUZCA COEFICIENTES
? 1, -4, 3


EL VALOR DE LOS COEFICIENTES ES
A= 1              B=-4              C= 3

SOLUCIONES
X1= 3
X2= 1
```

Si se desea resolver de nuevo el problema sin necesidad de volver a ejecutar el programa, veremos en la lección siguiente cómo conseguirlo mediante la sentencia GOTO.

PREGUNTAS Y PROGRAMAS DE REPASO

1. ¿Cuál es la función de la sentencia LET?
2. ¿Qué significa el término ASIGNACION en programación?
3. Indicar el contenido de cada variable después de la ejecución de estas cinco sentencias:

```
10 LET N=7
20 LET P6=17.5
30 LET Q=N+P6
40 LET N=N+10
50 LET Z=P6-N
```

4. Localizar los errores que existen en el siguiente programa:

```
200 LET M=5
220 LET V5=3.5
230 LEM T=6.5
240 LET 1AB=4
250 LET P=4.31
```

5. ¿Dónde se sitúa la sentencia END?
6. ¿Cuál es el error de la sentencia 100 PRINT X: Y: X?
7. ¿Cuál es la misión de la coma y punto y coma en una sentencia PRINT?
8. ¿Es correcta la siguiente sentencia PRINT?:

```
500 PRINT R, H*T, 3+5
```

9. Estudiar el siguiente programa:

```
10 I=15
20 P=37
30 PRINT P+I
40 M=5
50 P=8
60 PRINT P+I, M
70 END
```

   ¿Qué va a imprimirse en las líneas 30, 60?

10. Escribir un programa que realice el cálculo y la impresión de los resultados correspondientes a las expresiones siguientes:

```
A=3.5+10.4
B=3.5+5.2
C=6.4+5.2+9.5
```

   Cree variables para los valores 3.5, 6.4 y 5.2.

11. Escribir un programa que realice el cálculo y la impresión de los resultados correspondientes a las expresiones siguientes:

```
4.8+4.9-7.6=
1.4-6.6+5.8=
4.7×4.5×7.6=
```

No utilizar variables para los valores y el resultado sino solamente la sentencia PRINT.

12. ¿Cuál es el resultado de la ejecución?:

```
10 PRINT "ESTO ES UN EJEMPLO"
20 END
```

13. ¿Cuál es el resultado del programa?:

```
10 X = 5
20 Y = 7
30 Z = X*Y
40 Y = 9
50 PRINT X, Y, Z
60 END
```

14. Escribir un programa que imprime 20 asteriscos, a partir de la columna 10.

15. Escribir un programa que imprima este dibujo:



CAPITULO 4

Sentencias de control

O. INTRODUCCION

A lo largo de esta lección se tratarán las sentencias que permiten instruir a la computadora sobre transferencias de control en un programa. La posibilidad de saltar de una parte de un programa a otra es una característica esencial de la computadora. Estas transferencias de control permiten la posibilidad de tomar decisiones lógicas de acuerdo con el/los resultado/s de las operaciones parciales.

Al finalizar esta lección podrá utilizar todas las sentencias de transferencias de control de modo que aprenderá a manejarlas correctamente, permitiendo que el tratamiento de datos y la gestión de cálculos se flexibilice considerablemente.

Las sentencias que se examinarán son:

```
GOTO
IF - THEN
IF - THEN - ELSE
FOR - NEXT
ON - GOTO
WHILE - WEND
UNTIL - NEXT
```

1. CLASIFICACION DE LAS SENTENCIAS DE CONTROL

Las instrucciones en BASIC de un programa se ejecutan en orden creciente; sin embargo, existen unas sentencias de control que permiten romper o interrumpir la secuencia ascendente, a voluntad del programador. Estas sentencias se denominan de *transferencias de control* o *bifurcaciones* y se clasifican en dos tipos: *condicionales* e *incondicionales*.

Una sentencia de control «incondicional» realiza un salto sencillo o único a un número de línea. Son sentencias características incondicionales: GOTO, ON GOTO y GOSUB, ON GOSUB (tratamiento de subrutinas). Estas sentencias se realizan siempre, «incondicional», con independencia del desarrollo del programa.

Una sentencia de control «condicional» realizan salto, cuya ejecución depende del desarrollo precedente: son sentencias características condicionales IF-THEN, IF-THEN- ELSE, FOR-NEXT, UNTIL, WHILE.

Dentro de las sentencias de control condicional, conviene resaltar IF-THEN-ELSE, WHILE y WEND, REPEAT-UNTIL o UNTIL-NEXT, que permiten al usuario realizar sus programas con alguna de las técnicas conocidas como programación estructurada. Estas técnicas descomponen el problema en módulos que se programan independientemente y que luego se ensamblan consiguiendo programar mas fiables, legibles y, por supuesto, estructurados; todo ello redundará en beneficios para la edición y puesta a punto de programa.

Otras versiones de BASIC admiten otras sentencias de programación estructurada más potentes, como son los casos de las microcomputadoras BBC, QL, Macintosh, etc.

Algunas versiones de BASIC que utilizan estas sentencias de control son, además de las mencionadas, M: MBASIC, MSX BASIC, GW BASIC, VAX BASIC, HP BASIC, Amstrad BASIC, etc.

Las sentencias GOSUB y ON GOSUB se tratarán en una lección específica de SUBRUTINAS.

Dada la importancia creciente que va adquiriendo la programación estructurada, dedicaremos párrafos completos al estudio de las mencionadas sentencias IF-THEN-ELSE, WHILE-WEND y UNTIL-NEXT.

2. GOTO

Esta sentencia permite la *bifurcación incondicional.* Se modifica la secuencia normal de ejecución del programa, realizándose la transferencia del control a la línea que indique la citada sentencia.

Formato: n GOTO x

n Número de línea de programa.
x Número de línea del programa a la que se desea saltar.

Ejemplos:

```
100 GOTO 170
```
Saltar a la línea 170 y a continuación seguir secuencialmente.

```
120 GOTO 300
```
Saltar a la línea 300 y a continuación seguir secuencialmente.

El salto se puede considerar positivo o negativo o, lo que es igual, un salto hacia adelante o hacia atrás.

Programas sencillos:

1) *Bucle sin fin*

```
100 INPUT A
120 B=A+5
140 PRINT B
160 GOTO 100
200 END

RUN
? 5
 10
? 10
 15
? 20
 25
...
...
```

Como se ve en el programa, éste transfiere siempre el control a la línea 100 cada vez que se ejecuta la línea 160. Sin embargo, se debe hacer notar en este caso que la computadora entra en un *bucle sin fin* (infinito), dado que siempre se está realizando el programa y no se detendrá nunca. Evidentemente este programa no es práctico y se debe evitar que un programa entre en un *bucle sin fin.*

Un *bucle** *(lazo)* de programa es la parte del mismo que se realiza repetidamente, durante la ejecución debida a una sentencia de control.

Veremos en el párrafo siguiente el procedimiento de evitar *bucles sin fin.*

2) *Ejecución continua*

Supongamos que el programa de aplicación del capítulo 3 se desea ejecutar un número determinado de veces, pero sin necesidad de usar en cada ocasión la orden RUN. Introduzca la línea 170 en el mencionado programa y ejecútelo:

```
170 GOTO 25
```

Tras la presentación de resultados la línea 170 transfiere el control del programa a la línea 25, por lo que el mismo se vuelve a ejecutar nuevamente y nos ahorramos tener que pulsar o teclear RUN a cada nueva ejecución.

NOTAS PRACTICAS

a) El control del programa se puede llevar a cualquier línea del mismo siempre que ésta exista. Si la sentencia GOTO manda a una línea de programa inexistente se producirá un error de programación.

b) Cuando un programa entra en un bucle sin fin sólo se puede salir del mismo mediante la pulsación (no se puede escribir) de la tecla STOP o BREAK.

c) Aunque la sentencia ON GOTO permite la transferencia condicional según un valor de una variable que sigue una ley determinada. Dicha sentencia se tratará como *incondicional* y así la consideraremos cuando se estudie posteriormente.

d) La sentencia GOTO es problemática, por lo que le recomendamos máxima precaución en su utilización, y que la evite siempre que pueda ser, sustituyéndola por IF- THEN, GOSUB, ON-GOSUB, etc.

3. IF-THEN

La *bifurcación condicional,* al contrario de la incondicional, permite la elección de dos caminos dentro de la ejecución del programa y en momentos predeterminados.

Para ejecutar la bifurcación «condicional» se debe hacer uso de los operadores de relación o lógicos:

=	*Igual*
< >	*Distinto a*
= >	*Igual o mayor que*
<	*Menor que*
= <	*Igual o menor que*
>	*Mayor que*
AND	
OR	
NOT	

Formato:

n IF *expresión de comparación* (si) THEN (entonces) *acción*

n	Número de línea.
expresión de comparación	Expresión definida mediante operadores de relación o lógicos.
acción	Operación a realizar si se cumplen las condiciones de la expresión de comparación y que puede adoptar diferentes formas: a, b, c, etc.

* En Hispanoamérica es frecuente utilizar el término *lazo* en lugar de bucle. Ambos términos son traducciones de la palabra inglesa *"loop"*

a)

```
100 IF A<B THEN 500
200 IF A>4 THEN 600
300 IF A=5 THEN 120
```

La línea 100 indica que «si» A es < B «entonces» se salta a la línea 500 y, en caso contrario, se sigue la ejecución en la línea siguiente más alta.

La línea 200 indica que «si» A es > 4 «entonces» se salta a ejecutar la línea 600 y si A es < o = B sigue la ejecución del programa en la línea siguiente.

La línea 300 indica que «si» A=5 «entonces» se salta a la línea 120 y si es distinto de 5 el programa sigue ejecutándose en línea siguiente.

En realidad, la sentencia IF THEN incluye implícitamente una sentencia GOTO:

```
500 IF A=5 THEN (GOTO) 400
```

b) La mayoría de las versiones de BASIC admiten el siguiente formato:

```
100 IF A<B THEN GOTO 500
200 IF A>4 THEN GOTO 600
300 IF A=5 THEN GOTO 120
```

Este formato es equivalente al del apartado *a).*

c) La *acción* puede ser la ejecución de sentencias o expresiones válidas en BASIC.

```
100 IF A=5 THEN PRINT "MORTIMER"
200 IF A<>20 THEN A=A+1
```

La línea 100 nos indica que «si» A=5 «entonces» imprimir en pantalla la palabra «MORTIMER»; en caso contrario, continuar la ejecución del programa.

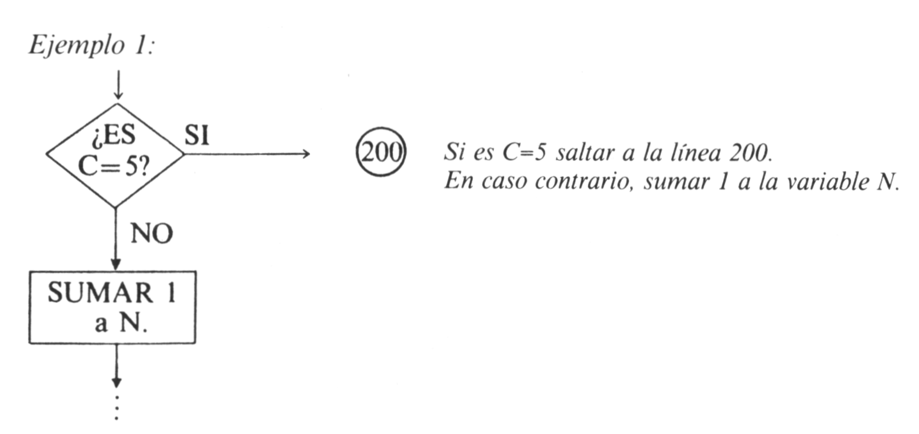
La línea 200 indica que «si» A es distinto de 20 «entonces» incrementar en 1 la variable A; en caso contrario, continuar la ejecución del programa.

d) Las expresiones de comparación pueden ser simples o complejas:

```
100 IF (A*B^2-4*A*C)/2^N<X*Y  THEN 400
```

e) La sentencia IF THEN se suele utilizar con la sentencia GOTO y una clave o etiqueta al final en el modo de trabajo llamado *bucle* o *lazo.* Como ya se ha definido, un BUCLE es un conjunto de operaciones que se repiten de un modo cíclico; en nuestro caso, mientras se cumple la expresión de comparación de la sentencia IF THEN. Se llama *iteración* a la realización del bucle, de modo que si un bucle se realiza *n* veces, se habrán realizado *n* iteraciones.

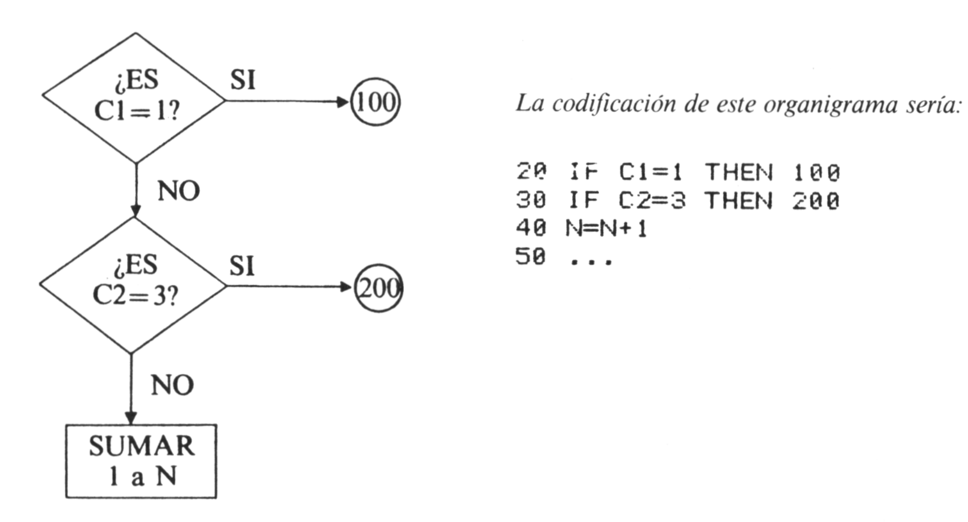
Ejemplo 1:



La codificación de esta parte de organigrama es:

```
100 IF C=5 THEN 200
120 N=N+1
...
...
200
```

Ejemplo 2:



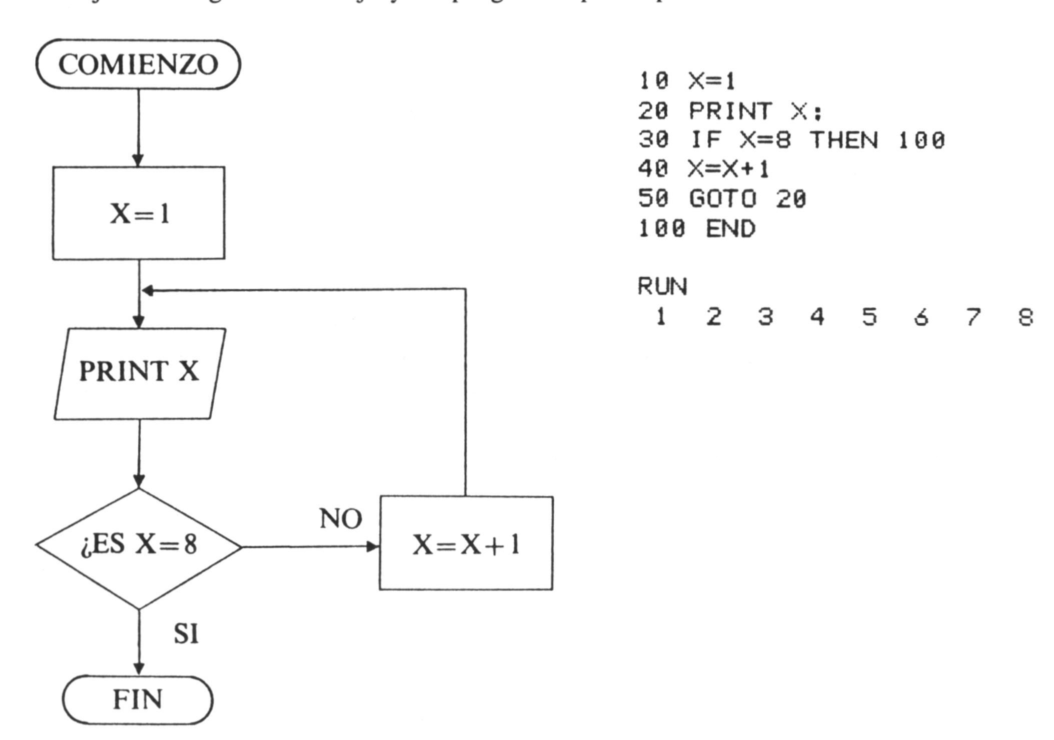
La codificación de este organigrama sería:

```
20 IF C1=1 THEN 100
30 IF C2=3 THEN 200
40 N=N+1
50 ...
```

La combinación de una sentencia IF THEN y una sentencia GOTO permite la realización de contadores, que es una de las facetas más interesantes en programación y que permitirá la realización de bucles en un número que vendrá definido por dichos contadores.

Ejemplos de programas:

1. Dibujar un diagrama de flujo y un programa que imprima los enteros del 1 al 8.




```
10 X=1
20 PRINT X:
30 IF X=8 THEN 100
40 X=X+1
50 GOTO 20
100 END

RUN
 1  2  3  4  5  6  7  8
```

En la línea 20 se imprime el entero 1; en la línea 30 se comprueba o pregunta si X=8, como en la primera iteración no sucede así, se ejecuta la línea 40, con lo que incrementa el valor de X en 1, yendo a continuación a la línea 20 e imprimiéndose en esta ocasión el entero 2. Al terminar de escribir 8 o, lo que es igual, después de la 8 iteración, en la línea 30 como X=8 se bifurcará a la línea 100 y se terminará el programa.

El programa se podría haber modificado de la forma siguiente:

```
10 X=1
20 PRINT X;
30 X=X+1
40 IF X>8 THEN 100
50 GOTO 20
100 END

RUN
 1  2  3  4  5  6  7  8
```

Como se ha visto, la comparación se puede hacer de dos formas diferentes, según que el contador N se realice antes o después de la sentencia IF THEN.

Un tercer método de realizar el mismo programa sería:

```
10 X=1
20 IF X>8 THEN 100
30 PRINT X;
40 X=X+1
50 GOTO 20
100 END

RUN
 1  2  3  4  5  6  7  8
```

2. Realizar un programa que sume cinco números cualesquiera.

```
10 INPUT "VALOR"; A
20 T=T+A
30 N=N+1
40 IF N=5 THEN 100
50 GOTO 10
100 PRINT "SUMA="; T
200 END
```

Le recordamos que al no inicializar las variables, a T y N se les asigna el valor 0. Si tiene problemas en su computadora haga 5 LET T=0: LET N=0.

Mediante la línea 10 se introducen los valores de A. La línea 40 realiza las sumas sucesivas. La línea 30 es el contador que cuenta los números introducidos. Las líneas 40 y 50 son las bifurcaciones que permiten se realice el proceso mientras el valor de N es menor que 5. El proceso sería el siguiente: 0+1, 1+2, (1+2)+3, (1+2+3)+4, (1+2+3+4)+5.

Ejecute el programa y vea el resultado si A es 5, 7, 9, 4, 8.

```
RUN
VALOR? 5
VALOR? 7
VALOR? 9
VALOR? 4
VALOR? 8
SUMA= 33
```

3. El siguiente programa calcula una tabla de dos columnas, en la primera se imprimen los números del 1 al 10 y en la segunda columna sus cuadrados:

```
10 REM CUADRADOS
20 PRINT "TABLA DE CUADRADOS"
30 PRINT
40 PRINT
50 X=1
60 PRINT X, X^2
70 X=X+1
80 IF X>10 THEN 200
90 GOTO 60
200 END

RUN


TABLA DE CUADRADOS


 1              1
 2              4
 3              9
 4              16
 5              25
 6              36
 7              49
 8              64
 9              81
 10             100
```

NOTAS PRACTICAS

a) *Sentencias IF ... THEN válidas son:*

`IF A=B THEN 50`	*El control se transfiere a la línea 50 sólo si A es igual a B.*
`IF A<B THEN 50`	*El control se transfiere a la línea 50 sólo si A es menor que B.*
`IF A<>B THEN 50`	*El control se transfiere a la línea 50 sólo si A es distinto de B.*
`IF A>B THEN 50`	*El control se tranfiere a la línea 50 sólo si A es mayor que B.*
`IF A<=B THEN 50`	*El control se transfiere a la línea 50 sólo si A es menor o igual que B.*
`IF A>=B THEN 50`	*El control se transfiere a la línea 50 sólo si A es mayor o igual que B.*
`IF A>B AND C>D THEN 400`	*El control se transfiere a la línea 400 sólo si se cumplen ambas condiciones: A > B y C > D.*

b) *Contadores.*

Una de las aplicaciones más utilizadas de la sentencia IF THEN es la formación de contadores que permita realizar cuentas de cualquier tipo.

1. Supongamos que se desea sumar los 150 primeros números enteros e imprimir su media. El procedimiento más sencillo sería realizar un contador que cuente hasta 150 y que vaya sumando en orden creciente dos números consecutivos, de modo que un sumando sea la suma parcial de los N−1 primeros números y el otro sumando sea el número N. El proceso

sería el siguiente: 0+1, 1+2, (1+2)+3, (1+2+3)+4, (1+2+3+4)+5, etc.; al llegar a 150 se detendría el proceso y se imprimiría la media.

Contador 1-150

```
100 N=0: S=0
110 N=N+1
120 IF N<=150 THEN 110
```

Sumador parcial o totalizador

```
100 N=0: S=0
110 N=N+1
120 S=S+N
130 IF N<=150 THEN 110
```

Media e impresión resultados

```
140 M=S/N
150 PRINT "MEDIA=": M
```

El programa total sería

```
100 N=0: S=0
110 N=N+1                    Contador
120 S=S+N                    Sumador
130 IF N<=150 THEN 110
140 M=S/N
150 PRINT "MEDIA="; M
200 END
```

La línea 100 en realidad no es necesaria, ya que la inicialización a 0 no se precisa. (Recuerde que una variable sin inicializar toma el valor 0, excepto en aquellas computadoras que así lo exijan, como ya conoce el lector.)

2. Programa para contar los números pares hasta 220 y su impresión.

```
100 N=0
120 N=N+2
140 PRINT N
160 IF N>220 THEN 200
180 GOTO 120
200 PRINT "FIN"
240 END
```

En este caso, se ha cambiado el orden del operador de relación y ha sido preciso incluir una sentencia GOTO que salte a la línea 120 mientras N sea menor que 220 o igual a 220.

El uso de contadores es de gran utilidad y aplicación en programación y es muy raro el programa que no incluya al menos algún tipo concreto.

c) *Empleo de sentencias IF- THEN dentro de una misma línea que ya contiene otras sentencias.*

Como ya sabemos, hay versiones de BASIC que admiten varias sentencias en una línea con tal de que se independicen mediante un *separador* (:), &, etc.

¿Qué ejecutaría la computadora al llegar a esta línea?

```
100 IF A<10 THEN 400: PRINT A, A^2, B*2: PRINT
120 PRINT "OK"
...
```

Si A es menor que 10, saltará a la línea 400. Si A es mayor o = que 10, pasará a ejecutar la línea 120, lo que significa que las sentencias PRINT A, A 2, B * 2: PRINT no se ejecutarán.

Por ello, si desea que se realicen esas sentencias habría de programar la línea 100 de la forma siguiente:

```
100 IF A<10 THEN PRINT A, A^2, B*2: GOTO 400
120 PRINT "OK"
...
```

3.1. Aplicaciones usuales de IF-THEN

a) Comprobación del valor de un número o variable numérica comprendido entre un límite inferior I y un límite superior S.

```
100 IF N<I  AND  N>S  THEN ...
```

b) Comprobación del valor de una variable de cadena o cadena de caracteres para verificar si una respuesta o valor calculado es correcta.

Ejemplo: Introducir por teclado una variable de cadena y comprobar si ha sido una de las dos respuestas esperadas.

```
10 INPUT A$
20 IF A$="PEPE" OR A$="MANOLO" THEN 50
30 PRINT "FIN"
40 STOP
50 PRINT "CORRECTO"
```

Si se introduce, al ejecutarse, el nombre «PEPE» o «MANOLO» se salta a la línea 50 y se imprime CORRECTO. En caso contrario se imprimirá FIN.

c) Comprobación de dos condiciones excluyentes y complementarias.

```
10 INPUT A$
20 IF NOT A$="S" AND NOT A$="N" THEN 10
30 PRINT "OK"
40 END
```

d) Bifurcaciones condicionales múltiples:

El programa siguiente comprueba dos números, detectando si son iguales o diferentes, y en este caso cuál de ellos es mayor.

```
10 INPUT A, B
20 IF A>B THEN PRINT A; "MAYOR QUE"; B
30 IF A<B THEN PRINT A; "MENOR QUE"; B
40 IF A=B THEN PRINT A; "IGUAL A";B
50 GOTO 10
100 END
```

e) Comprobación de respuesta correcta con mensaje de error en caso contrario.

```
10 REM OPCION DOBLE
20 PRINT "C CORRECTO"
30 PRINT "F FALSO"
40 INPUT A$
```

```
50 IF A$="C" THEN 400
60 IF A$="F" THEN 500
70 PRINT "ERROR"
80 PRINT "INTRODUZCA C/F"
90 GOTO 20
...
400 ...
.......
500 ...
.......
1000 END
```

Al ejecutarse el programa, si en la línea 40 se introduce C el control del programa salta a la línea 400; si se introduce F, salta el control a la línea 500; cualquier entrada distinta de C o F hace que se imprima.

```
ERROR
INTRODUZCA C/F
C CORRECTO
F FALSO
```

A continuación se volvería a introducir otro valor en la línea 40.

Este programa obliga al usuario a introducir C ó F, dado que en caso contrario *nunca* realizaría las líneas 400 ó 500.

f) Menú.

El ejemplo anterior es un caso típico de Menú. En terminología de programación, *Menú* es la presentación en pantalla de una serie de opciones al objeto de permitir al usuario la selección de una de ellas.

El Menú suele ser parte integrante de cualquier programa y formato indispensable en el tratamiento de ficheros.

Ejemplo:

```
10 REM DEMO
20 PRINT "INTRODUZCA 1, 2, 3";
30 INPUT A
40 IF A=1  THEN 150
50 IF A=2  THEN 200
60 IF A=3  THEN 300
70 PRINT "ELIJA DE NUEVO"
80 GOTO 20
150 PRINT "AZUL"
160 GOTO 20
200 PRINT "ROJO"
210 GOTO 20
300 PRINT "AMARILLO"
310 GOTO 20
500 END
```

Al ejecutarse el programa, si en la línea 30 se introduce, por ejemplo 2, el progama saltará la línea 200, se imprimirá ROJO y retornará el control a la línea 20.

Si se introduce 5, se ejecutan las líneas 40, 50, 60, 70 y 80, con lo que no se imprimirá nada en pantalla, retornando el control a la línea 20 hasta que se introduce 1, 2 o 3, que son los dos solicitados.

4. FOR-NEXT

El tratamiento de bucles o lazos es una de las técnicas más utilizadas en programación.

Un *bucle* o *lazo* es un conjunto de instrucciones que se ejecutan repetidamente, de tal modo que se varía alguno o varios elementos en cada ejecución. Las instrucciones se pueden ejecutar diez veces o más.

Una *iteración* es la realización del conjunto de instrucciones. Un bucle tiene *n* iteraciones si se ejecuta *n* veces.

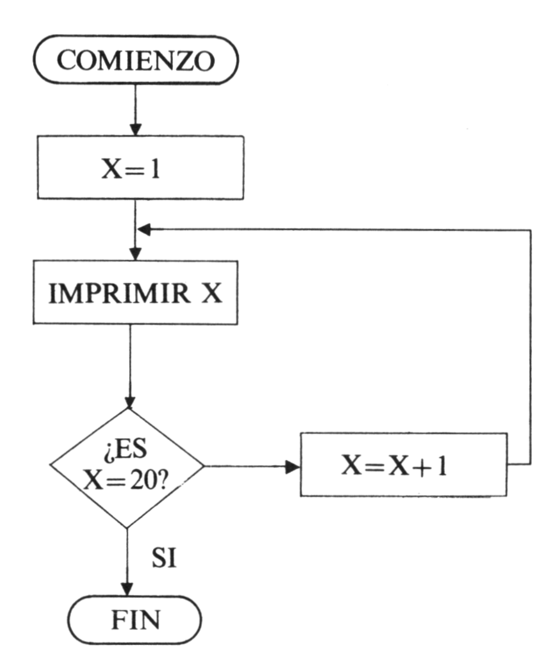
Los bucles han sido ya utiliados con las sentencias IF-THEN y GOTO.

Así, un bucle sencillo sería:

```
100 INPUT A
200 PRINT A
300 GOTO 100
400 END
```

Este programa imprimirá continuamente el número introducido en la línea 100, pero no se interrumpirá nunca o, lo que es igual, el bucle no tiene fin y se dice que el programa ha entrado en *un bucle sin fin,* lo que significa que *nunca* terminará su ejecución *ni* realizará la línea 400.

Ejemplo 1: Realizar un programa que imprima los enteros del 1 al 20 ambos inclusive. El organigrama podría ser el siguiente:



La codificación sería:

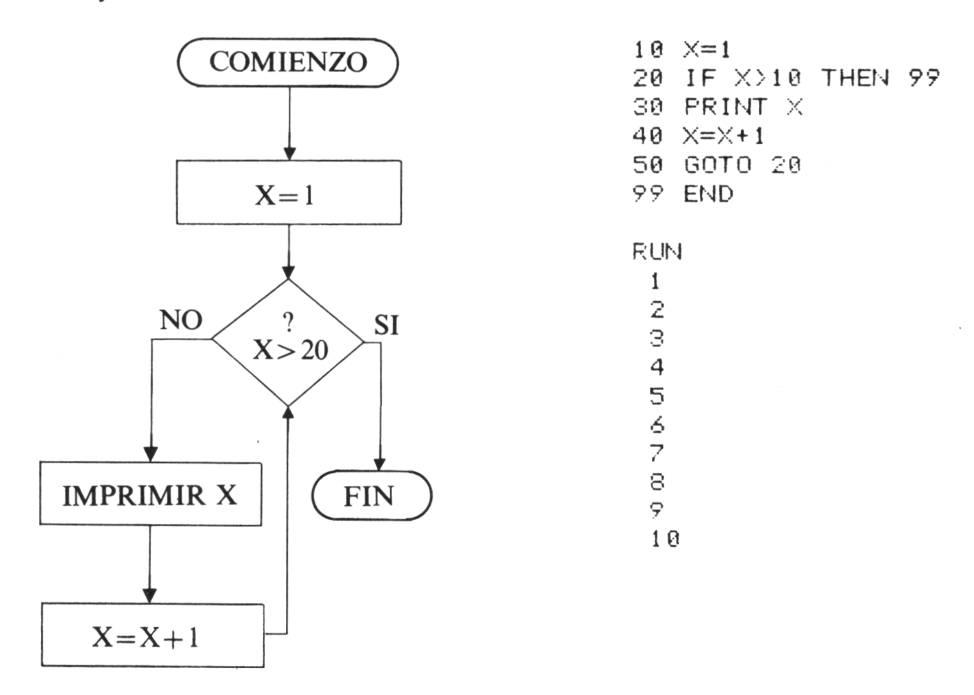
```
10 X=1
20 PRINT X
30 IF X=10 THEN 99
40 X=X+1
50 GOTO 20
99 END
```

Si se ejecuta con RUN:

```
1
2
3
4
5
6
7
8
9
10
```

El programa repite la ejecución de varias instrucciones, 20, 30, 40, 50, hasta que el valor indicado para X (20 en este caso) sea detectado o alcanzado, en cuyo instante se termina el bucle.

Ejemplo 2: El programa y organigrama anterior se puede realizar también de este modo. La ejecución y resultados se muestran en la columna de la derecha.



Bucles automáticos: Los bucles vistos anteriormente repiten la secuencia completa de instrucciones, hasta que se cumple una cierta condición o se han realizado un cierto número de bucles, antes de proseguir con el resto del programa. Sin embargo, se puede conseguir que la realización de los bucles sea automática, con sólo indicr el valor inicial y final de los mismos.

Los bucles automáticos se obtienen mediante la sentencia FOR-NEXT.

El *formato* de la sentencia FOR- NEXT es:

n FOR *variable=relación* o *expresión*$_1$ TO *expresión*$_2$ STEP *expresión*$_3$

n	Número de línea de programa.
variable	Nombre de la variable que actúa como índice para evaluar el número de veces que se realizará el bucle.
relación o expresión$_1$	Valor inicial de la variable.
expresión$_2$	Valor final de la variable que no se puede superar.
expresión$_3$	Valor que toma el incremento de la variable cada vez que se alcanza la sentencia NEXT.

Ejemplos de sentencias FOR-NEXT válidas

```
100 FOR I=1     TO 100
120 FOR I=A     TO B
140 FOR J=5*N   TO C^2  STEP 4
200 FOR I=5     TO 50
300 FOR I=20    TO 5    STEP-1
400 FOR I=100   TO 1    STEP-3
500 FOR U=N     TO W*R  STEP (H+L)/B
```

La sentencia FOR da un valor inicial a un conductor y comprueba el fin del bucle. La sentencia NEXT aumenta el contador indicado por la sentencia FOR.

Un ejemplo sencillo puede ser:

```
10 FOR I=1  TO 5
20 PRINT "OK"
30 NEXT I
40 END
```

La línea 10 establece los parámetros del bucle, en nuestro caso *valor inicial*=1 y *valor final*=10.

Cuando se ejecuta el programa (tecleando RUN y RETURN) el valor inicial de I es 1, se oprime OK y al llegar a la línea 30 el contador I del bucle se incrementa en una unidad y el programa salta a la línea 10, en cuyo instante se compara este valor 2 con el valor final 10. Mientras *valor actual< valor final* el programa salta a la línea siguiente, a la sentencia FOR, y se volverá a realizar el bucle, incrementándose el valor de I en cada iteración hasta que se cumpla *valor actual> valor final,* en cuyo momento el programa salta a la instrucción siguiente, inmediatamente a la línea que contiene la sentencia NEXT, en nuestro caso 40.

En nuestro ejemplo el bucle se realia diez veces.

Obsérvese que *un bucle FOR-NEXT se ejecutará como mínimo una vez.*

Así pues, el programa ejemplo, suponiendo que ahora el valor final es 5, al ejecutarse imprimirá:

```
OK
OK
OK
OK
OK
```

Ejemplo 3: Los ejemplos 1 y 2 anteriores que realizan la impresión de los *n* primeros números enteros se puede realizar mediante la sentencia FOR-NEXT del modo siguiente:

```
10 FOR X=1 TO 20
20 PRINT X
30 NEXT X
40 END

RUN
 1
 2
 3
 .
 .
 .
 20
```

Como se ve, en este ejemplo se han eliminado dos líneas de programa, con lo que se puede comprobar en principio el ahorro que supone la utilización de esta sentencia.

Ejemplo 4:

```
10 FOR I=1  TO 5
20 PRINT "*"
30 NEXT I
40 END
```

En este programa, I representa el contador y varía de 1 a 5 en incrementos de 1. El programa imprime «*» cinco veces. El bucle comprende tres instrucciones, comienza con la sentencia FOR I y termina con la sentencia NEXT I.

Ejemplo 5:

```
10 S=0
20 FOR I=1  TO 5
30 S=S+I
40 NEXT I
50 PRINT S
60 END

RUN
 15
```

Este programa obtiene la suma de los enteros comprendidos entre 1 y 5. Se inicializa en cero y en cada iteración toma los valores sucesivos 1,1+2, 1+2+3, 1+2+3+4, 1+2+3+4+5.

Modificando la línea 20 se podría obtener la suma de valores comprendidos entre 1 y 500, por ejemplo:

```
10 S=0
20 FOR I=1  TO 500
30 S=S+I
40 NEXT I
50 PRINT S
60 END

RUN
 125250
```

Ejemplo 6: El bucle no tiene que comenzar en 1, pudiendo ser el valor inicial 1 o distinto de 1, o una expresión que represente cualquier valor numérico. De igual modo, el valor final no tiene límite físico apreciable ni tiene por qué ser un valor numérico concreto.

```
50 S=0
60 FOR I=2 TO 1000
70 S=S+I
80 NEXT I
90 PRINT S
100 END

RUN
 500499

50 A=5
100 B=20
120 FOR I=A  TO B
140 PRINT I
160 NEXT I
200 END
```

```
RUN
 5
 6
 7
 .
 .
 .
 20
```

Ejemplo 7: El valor del incremento (*expresión*$_3$ que sigue a la *expresión*$_2$) de la variable puede tomar cualquier valor.

```
100 FOR I=1  TO 1000 STEP 2
120 S=S+I
140 NEXT I
160 PRINT S
200 END

RUN
 250000
```

Ejemplo 8:

```
100 FOR I=43.4  TO 54.5 STEP 2
120 S=S+I
130 NEXT I
140 PRINT S
150 PRINT I
```

Los valores sucesivos de I son 43.4, 45.4 ..., 53,4.

```
RUN
 290.4
 55.4
```

Ejemplo 9: El incremento (STEP) puede ser negativo. En este caso el valor inicial del contador debe ser mayor que el valor final.

```
10 FOR C=10 TO 1 STEP-1
20 PRINT C;
30 NEXT C
40 END

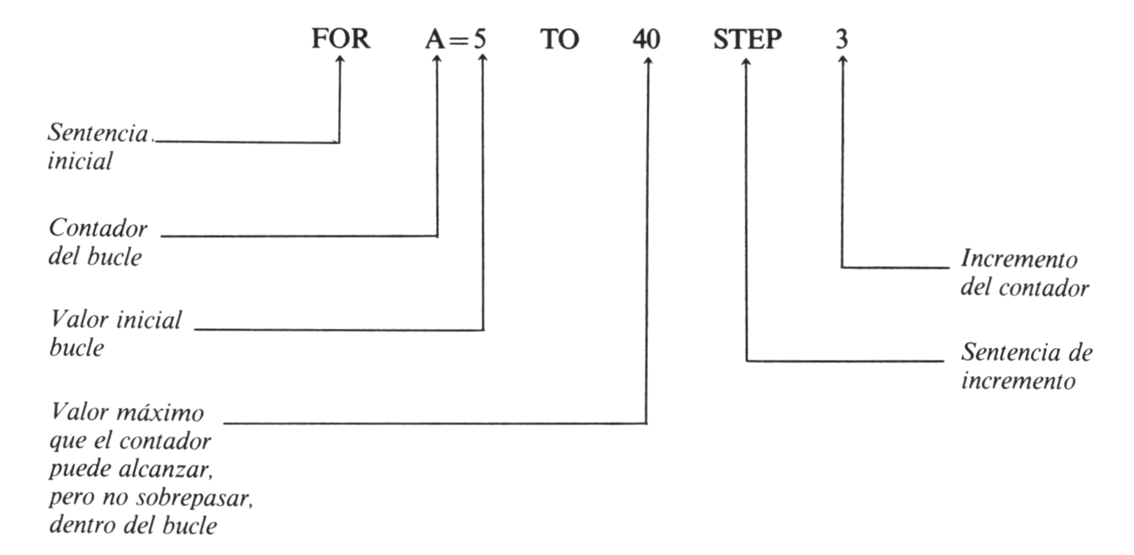
RUN
 10  9  8  7  6  5  4  3  2  1
```

Ejemplo 10: El valor inicial del contador puede ser negativo.

```
10 FOR I=-10  TO  100  STEP 5
20 S=S+I
30 NEXT I
40 PRINT S
50 PRINT I
60 END

RUN
 1035
 105
```

Sentencia FOR-NEXT típica — *FORMATO*



NOTAS:

1. Cuando no exista ambigüedad en el bucle se puede suprimir la variable de control de la sentencia (palabra) NEXT.

```
100 FOR I=1 TO 10                              100 FOR I=1   TO 10
120 PRINT I                                    120 PRINT I
140 NEXT                    equivalente a      140 NEXT I
160 END                                        160 END
```

No obstante, debe tener cuidado con esta simplificación, dado que algunas computadoras como HEWLETT PACKARD no admiten la omisión de la variable.

2. Si el incremento del contador PASO (STEP) es *positivo*, el valor inicial de la variable debe ser inferior o igual al valor final. Inversamente, si el PASO es *negativo* es necesario que el valor inicial sea superior o igual a dicho valor final.

3. Si no se cumple la hipótesis anterior el bucle sólo se ejecutará una vez.

Ejemplo: Escriba y ejecute el siguiente programa.

```
10 FOR I=10  TO  1   STEP 1
20 PRINT I
30 NEXT I
40 END

RUN
 10
```

Al ejecutarse el programa, en la primera iteración el valor de la variable I es 10 y al llegar a la línea 30 se comprueba que I no puede variar, por lo que se termina el bucle.

4.1. Bucles anidados

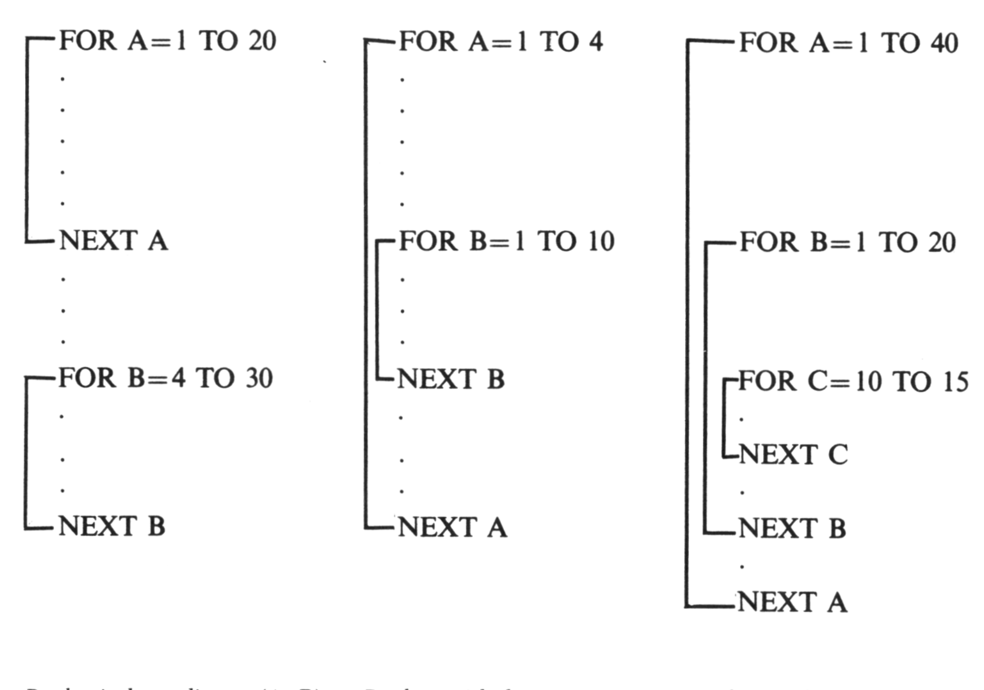
Una característica notable de la sentencia FOR-NEXT es la posibilidad de utiliar *bucles anidados.* Dos o más bucles se llaman *anidados* cuando cada bucle completo es interno a otro bucle y éste último interno a otro, y así sucesivamente.

Mediante los bucles anidados se podrán tratar listas y tablas de datos, así como el tratamiento de cálculos complejos.

Los bucles exigen para anidarse que sean escritos correctamente o, lo que es igual, que estén internos los unos en los otros.

En los dos párrafos siguientes se muestran estructuras de bucles anidados correctos y no correctos.

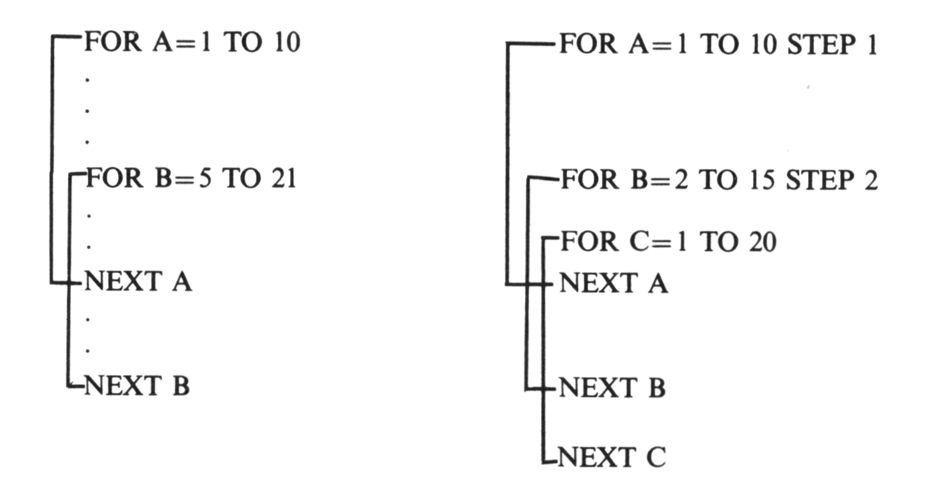
a) *Formatos correctos de bucles*



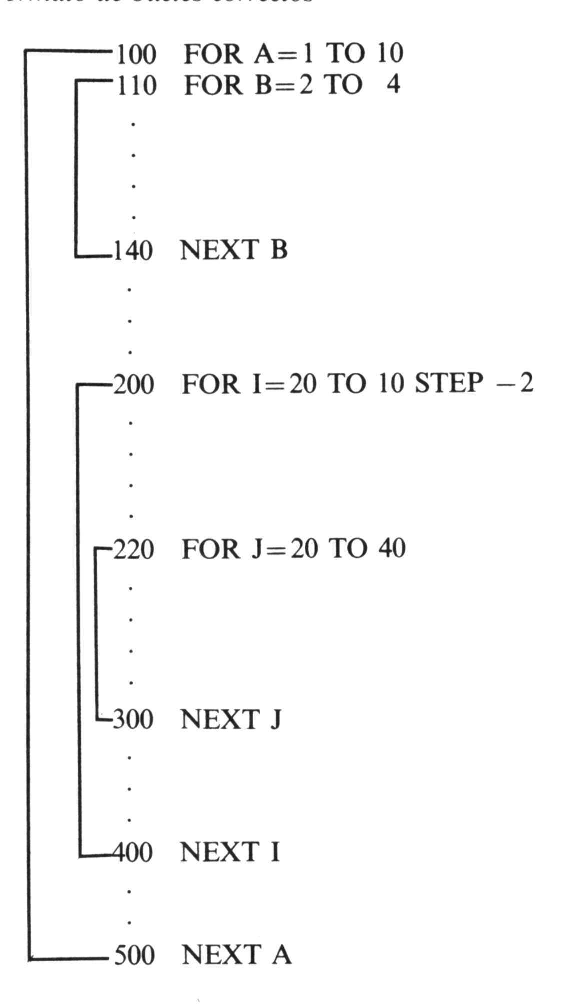
Bucles independientes (A, B) *Bucles anidados* (A, B) *Bucles anidados* (A, B, C)

b) *Formatos no correctos de bucles*

Los bucles internos no se pueden cruzar con los externos.



c) *Formato de bucles correctos*



Ejemplo práctico

Realice el siguiente programa y ejecútelo.

```
10 A$="X"
20 B$="Y"
30 FOR  I=1  TO 5
40 C$=C$+A$
50 FOR  J=1  TO 2
60 C$=C$+B$
70 PRINT C$
80 NEXT J
90 NEXT I
100 END


RUN
XY
XYY
XYYXY
XYYXYY
XYYXYY
.........
.........
```

NOTA: *Existen versiones de* BASIC *que cuando las sentencias* NEXT J, NEXT I, *etc., pertenecientes a bucles anidados, son contiguas y no existen líneas intercaladas entre ellas, se puede utilizar una sentencia con el siguiente formato* NEXT J, I, *etc. en lugar de ellas.*
Así, en el programa anterior las líneas 80 y 90 se podrían sustituir por una única línea 80.

```
80 NEXT J,I
```

Como norma general a lo largo del curso y a fin de evitar posibles errores se utilizarán notaciones independientes. (Si escribe NEXT I, J en lugar de NEXT J, I obtendrá el mensaje de error *NEXT whitout FOR in 40.)*

```
80 NEXT J
90 NEXT I
```

4.2. Salidas y entradas a bucles

En un programa se puede salir de un bucle en cualquier posición (mediante una sentencia IF-THEN, por ejemplo) antes de que el contador haya tomado todos los valores previstos.

```
10 A=0
20 FOR  I=1  TO 10
30 INPUT N
40 IF N=0  THEN  70
50 IF N>A  THEN  A=N
60 NEXT I
70 PRINT "MAXIMO"
80 PRINT A
90 END

RUN
? 5
? 7
? 3
? 0
MAXIMO
 7
```

Es preciso entrar al bucle por «la puerta principal», es decir, la sentencia FOR, y *no* por el *interior* de un bucle, ya que en este caso se provoca un error en el primer NEXT que se encuentre, ya que va a incrementar un contador no actualizado o el de otro bucle.
Una sentencia GOTO puede reenviar el control a una sentencia NEXT, con lo que se termina el bucle en curso y se pasa al siguiente (o a la sentencia siguiente si el contador ha alcanzado el límite fijado).

REGLA:

No se debe entrar en el bucle FOR-NEXT desde una sentencia externa más que por la sentencia FOR. Sin embargo, se puede salir en cualquier lugar del bucle.
El envío externo a una sentencia NEXT termina el bucle.

4.3. Obtención de retardos mediante bucles

En numerosas ocasiones es necesario realizar «temporizaciones» que produzcan la detención o el retardo de una parte del programa, o bien de una presentación de resultados.
Con sentencias STOP el programa se detiene un tiempo indefinido; sin embargo, es muy frecuente que la detención exija un tiempo determinado de tal modo que el programa continúe automáticamente al pasar o transcurrir dicho tiempo.

En muchas aplicaciones se necesita que un mensaje en pantalla permanezca un cierto tiempo, para permitir cómodamente su visualización, y a continuación, desaparezca de pantalla.

Las computadoras HEWLETT PACKARD (HP-9845) poseen una sentencia WAIT *n,* en la que *n* es el número de milisegundos que dura la pausa o detención del programa. Sin embargo, lo normal es que la mayoría de las computadoras no posean esta sentencia y haya que recurrir a algún «truco» de programación para conseguir la temporización.

El procedimiento utilizado para conseguir «retardos de tiempo» o «temporizaciones» es el empleo de un bucle vacío.

Un *bucle vacío* es un bucle que no contiene sentencias en su interior.

```
100 FOR I=1  TO  1000
120 NEXT I
```

Al ejecutarse un bucle vacío *sólo* se incrementa el contador en cada iteración o recorrido del mismo, de tal modo que en ejecutarse el bucle completo se tardaría $1000 \times t_p$, siendo t_p el tiempo que la computadora tarda en realizar una iteración o un recorrido.

Así, es usual que los microprocesadores internos de la computadora empleen 1 ms. (milisegundo) aproximadamente en una iteración (aunque no se sabrá exactamente si el fabricante no proporciona dicho tiempo). En el ejemplo anterior, el retardo ocasionado sería, por tanto, 1000×1 ms =1 segundo.

Algunas computadoras, por ejemplo COMMODORE (Commodore 64, Commodore 128, etc.), tienen un reloj de tiempo interno que se puede programar para que cuente horas, minutos y segundos mediante la sentencia TI$.

Así, su formato es TI$=HHMMSS, en donde HHMMSS es la hora inicial de puesta en marcha del reloj, expresada en horas, HH, minutos, MM, y segundos, SS. Al ejecutarse esta sentencia se pondría en marcha el reloj.

Si se deseara saber el tiempo que tardará la computadora en ejecutar un bucle vacío, bastaría introducir y ejecutar el siguiente programa:

```
10 TI$="000000": REM PUESTA EN MARCHA RELOJ
20 FOR I=1 TO 10000
30 NEXT I
40 PRINT TI$ : REM IMPRESION DURACCION 10000 ITERACIONES.
50 END

RUN
000011
```

El resultado indica que se ha tardado 000011 s. en 10000 iteraciones o, lo que es igual, 0000.11 s. en un bucle vacío.

Si consideramos que la duración de la ejecución de un bucle vacío es 1 ms., para obtener retardos de 1 s. 5 o 40 s. necesitará los siguientes bucles:

```
100 FOR I=1 TO 1000
200 NEXT I

100 FOR I=1 TO 5000
200 NEXT I

100 FOR I=1 TO 40000
200 NEXT I
```

Aplicación

Introduzca y ejecute el siguiente programa en su computadora:

```
10 FOR I=1  TO  1000 STEP 2
20 PRINT I
30 NEXT I
40 END

RUN
 1
 3
 5
 .
 .
 .
```

Al ejecutar este programa, los números se presentan en la pantalla a gran velocidad, de modo que es *prácticamente* imposible su visualización. Si entre cada impresión se introduce un retardo adecuado, la presentación de los resultados se ralentizará y se podrá leer cómodamente:

```
10 FOR I=1 TO 1000 STEP 2
20 PRINT I
30 FOR J=1 TO 1000
40 NEXT J
50 NEXT I
100 END
```

4.4. Ejercicios prácticos de programación

Practique los siguientes ejercicios, primero sobre papel y luego sobre su computadora:

Cuadrados de los números comprendidos entre 1 y 5.

```
10 FOR I=1 TO 5
20 PRINT I*I
30 NEXT I

RUN
 1
 4
 9
 16
 25
```

Cuadrados de los N primeros números.

```
10 INPUT "CUANTOS NUMEROS"; N
20 FOR I=1 TO N
30 PRINT I*I
40 NEXT I

RUN
CUANTOS NUMEROS? 8
1
4
9
16
25
36
49
64
```

Qué valores toma el índice I en los programas siguientes tras su ejecución.

a)

```
10 FOR I=1 TO 10 STEP 2        RUN
20 PRINT I;                    1   3   5   7   9
30 NEXT I
```

b)

```
10 FOR I=3 TO 17 STEP 5
20 PRINT I;                    RUN
30 NEXT I                      3   8   13
```

c)

```
10 FOR I=10 TO 1 STEP-3
20 PRINT I;                    RUN
30 NEXT I                      10   7   4   1
```

d)

```
10 FOR I=2 TO 12 STEP 3
20 PRINT I;                    RUN
30 NEXT I                      2   5   8   11
```

e)

```
10 INPUT A, B, C               RUN
20 FOR I=A  TO  B  STEP C      ? 5, 20, 4
25 PRINT I;
30 NEXT I                      5   9   13   17
```

f)

```
10 M=10
20 N=4                         RUN
30 FOR I=M/2  TO  N*N STEP N/2 5   7   9   11   13   15
40 PRINT I;
50 NEXT I
```

5. IF-THEN-ELSE

Hoy día, en muchas versiones de BASIC: MICROSOFT, (MBASIC) HEWLETT PACKCARD, Amstrad, MSX, BASICA, GW BASIC, CBASIC, etc., admiten una sentencia de bifurcación condicional de mayor potencia que la sentencia IF-THEN clásica ya tratada: IF-THEN-ELSE. Mediante esta sentencia se pueden realizar acciones adecuadas cualquiera que sea el resultado de una comparación. Además, la misma permite poner varias sentencias después de ELSE (inclusive otra sentencia IF ... THEN ... ELSE); en cambio, después de THEN sólo puede ir una sentencia que no sea IF.

El uso de la sentencia ELSE permite indicar una acción alternativa en caso de que la prueba o condición que viene tras IF sea falsa. Si no se usara ELSE, en el caso mencionado, el control pasa a la siguiente línea del programa.

Ejemplos:

```
10 INPUT A$
20 IF A$="SI" THEN 100 ELSE END
100 PRINT "OK"
```

En la línea 10 se introduce A$ si es igual a «SI», entonces el programa salta a la línea 100; pero si A$ no es «SI», el programa salta sobre la sentencia THEN y pasa a ELSE, la cual ordena parar el programa mediante END.

```
RUN
? SI
OK

RUN
? NO
READY
```
Fin del programa

Formato:

n IF *expresión* THEN *acción*$_1$ ELSE *acción*$_2$

n	Número de línea.
acción$_1$	Sentencias o número de línea que se realizan si el resultado de la operación lógica/aritmética es «CIERTA» o distinta de 0 (cero).
acción$_2$	Estas sentencias o número de línea se ejecutan si el valor de la expresión es FALSA o 0.

La sentencia IF comprueba la expresión y opera con valores lógicos. Si la comparación genera un valor lógico CIERTO (verdadero), la sentencia o sentencias que siguen a THEN se ejecutarán si la comparación resulta FALSA, entonces se ejecutarán la sentencia(s) que siguen a ELSE.

a)
```
100 IF  I=10  THEN 200  ELSE 400
```

Esta línea comprueba el valor de la variable I. Si es 10 se bifurca a la línea 200; en caso contrario, a la línea 400.

Esta línea equivale a:

```
100 IF  I=10  THEN  200
200 GOTO 400
```

b)
```
100 IF A<B THEN PRINT "A<B" ELSE PRINT "B>=A"
```

Si A es menor que B, la computadora imprime «A < B» y después prosigue con la línea siguiente saltando la sentencia ELSE. Si A es mayor que B, la computadora salta directamente a la sentencia ELSE e imprime «B > =A». Después pasa el control a la sentencia siguiente del programa.

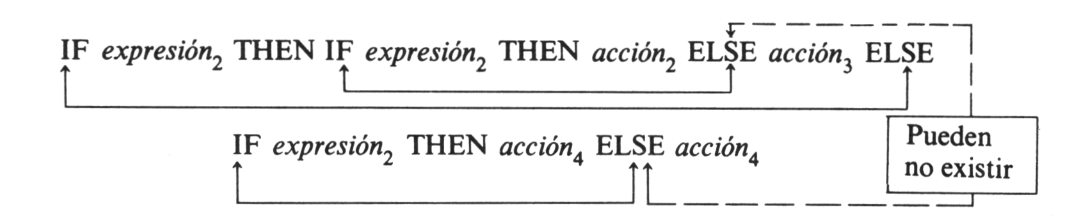
c)
```
200 IF A>.005  THEN  B=1/A: A=A/6 ELSE 400
```

Si A > .005 es CIERTO, entones se ejecutarán las dos sentencias asignando valores a B y a A. El programa pasa a continuación a la línea siguiente, saltándose la sentencia ELSE. Sin embargo, si A < =.005 es FALSA, el control pasa directamente a la sentencia ELSE que transfiere a su vez el control a la línea 400.

No se necesita GOTO después de ELSE para saltar a una línea concreta.

5.1. IF-THEN-ELSE anidados

Los bucles IF-THEN-ELSE se pueden *anidar* de igual modo que los bucles FOR-NEXT, con la única condición de que cada IF THEN debe corresponder con su ELSE.



Ejemplos:

a) IF X> Y THEN PRINT "MAS GRANDE" ELSE IF Y<X THEN PRINT "MAS PEQUEÑO" ELSE PRINT "IGUAL".

b) IF X<>Y THEN IF X<Y THEN PRINT "X<Y" ELSE PRINT "X>Y" ELSE PRINT "X=Y"

NOTA: *Si no existe el mismo número de THEN y de ELSE, cada ELSE se conecta con el THEN más próximo.*

IF A=B THEN IF B=C THEN PRINT "A=C" ELSE PRINT "A<>C"

La instrucción anterior significa que si A es diferente de B se pasa a la instrucción siguiente y no se ejecuta PRINT «A=C».

100 IF A>B THEN PRINT "A>B" ELSE
IF A<B THEN PRINT "A<B" ELSE PRINT "A=B"

```
100 IF A>B THEN PRINT "A>B"
        ELSE IF A<B THEN PRINT "A<B"
               ELSE PRINT "A=B"
```

Ejemplos:

Analicemos varios ejemplos de programas de aplicación.

a)
```
100 REM CALCULO MAYOR DE DOS NUMEROS
120 INPUT "INTRODUCIR 2 NUMEROS A, B"; A, B
140 IF A<=B THEN PRINT "A<=B" ELSE PRINT "A>B"
160 GOTO 100
180 END

RUN
INTRODUCIR 2 NUMEROS A, B? 72,46
A>B
```

Este programa calcula cuál es el mayor de dos números.

b)
```
100 REM DETECCION DE UN LIMITE DEFINIDO
120 A=1
140 IF A<10 THEN 200 ELSE 260
160 PRINT "NO PASO POR AQUI"
```

```
180 GOTO 299
200 PRINT A;
220 A=A+1
240 GOTO 140
260 PRINT "HALLADO LIMITE"
299 END

RUN
 1  2  3  4  5  6  7  8  9 HALLADO LIMITE
```

5.2. Resumen sentencia IF-THEN-ELSE

IF *expresión*$_1$ THEN *acción*$_1$ ELSE *acción*$_2$

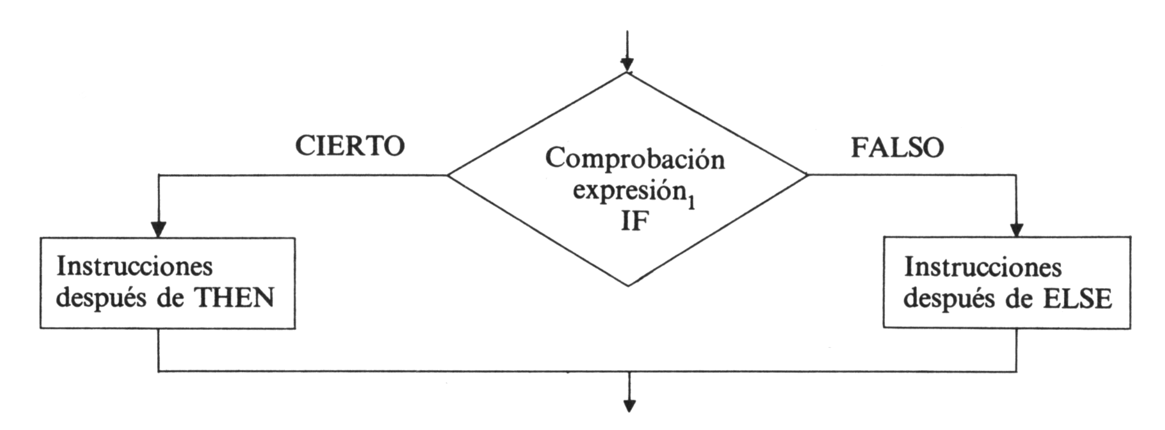
(SI) *(ENTONCES)* *(SI NO)*

Comprueba o verifica la expresión$_1$:

— Si es CIERTA (verdadera), se ejecutan la(s) sentencia(s)/o salta a la línea definida después de la sentencia THEN.
— Si es FALSA, se ejecutan las instrucciones que vienen después de ELSE.

Su desarrollo es el siguiente:

— La prueba (test) de la expresión$_1$ entega un 0 si es FALSA y −1 si es CIERTA.
— Si la respuesta es 0, se ejecuta la acción$_2$ (después de ELSE). Si no, se ejecuta acción$_1$ (después de THEN).




```
100 IF A THEN PRINT "A DISTINTO DE CERO" ELSE PRINT "A ES
IGUAL A CERO"

200 IF (A<15) AND (A>5) THEN PRINT "5<A<15"
```

En este caso, si $A < 15$ y $A > 5$ son «ciertas» se les asigna el valor −1 y en caso contrario 0.

NOTA: *En aquellas computadoras en que no se disponga de la sentencia ELSE se puede simular por las siguientes líneas:*

```
100 IF A=10  THEN  200
120 GOTO 400
```

equivalen a

```
100 IF A=10 THEN 200 ELSE 400
```

6. ON-GOTO

La sentencia ON-GOTO es una bifurcación múltiple condicional que incorpora un número de sentencias IF-THEN implícitas en una única sentencia. Al ejecutarse comprueba el valor de una expresión o variable y en función del mismo bifurca a los números de líneas que van a continuación de GOTO.

Formato:

n ON *expresión* GOTO n_1, n_2, n_3, n_4...

n	Número de línea.
expresión	Expresión lógica/aritmética o variable legal.
n_1, n_2, n_3	Números de líneas.

Al ejecutarse esta sentencia, la computadora evalúa la variable o expresión, y si es un número real lo convierte mediante truncamiento en un entero tal como I... de tal modo que el control del programa se transfiere a la línea 1.ª que viene tras GOTO.

```
100 ON  A  GOTO 400, 500, 200, 600
```

Al ejecutarse la línea 100 se realizará lo siguiente:

Si A es 1, se bifurca a la línea 400.
Si A es 2, se bifurca a la línea 500.
Si A es 3, se bifurca a la línea 200.
Si A es 4, se bifurca a la línea 600.

Esta sentencia es equivalente a las cuatro siguientes:

```
10 IF A=1  THEN 400
20 IF A=2  THEN 500
30 IF A=3  THEN 200
40 IF A=4  THEN 600
```

NOTAS:

a) Si el valor de la variable/expresión es 0 o mayor que el número de líneas, se ejecuta la sentencia que sigue a ON, o sea la línea siguiente, o bien produce error en algunos modelos de computadoras. (En el ejemplo de la línea 100, si A vale 0.5 o mayor que 5).

b) Si el valor de la variable/expresión no es entero, la computadora trunca automáticamente su valor y lo convierte en entero.

c) A una sentencia ON-GOTO le pueden seguir tantos números de línea como quepan en una línea.

d) Se puede utilizar varias veces el mismo número de línea, así:

```
ON A GOTO 400, 500, 400, 600, 500
```

prosigue la ejecución es:

— 400 si A es 1 ó 3
— 500 si A es 2 ó 5
— 600 si A es 4

Ejemplos de aplicación

1. Esta sentencia se suele utilizar con gran profusión, sobre todo cuando se ha de seleccionar un «menú». Recordemos que un menú ofrece al usuario, por pantalla, la posibilidad de elección entre varias opciones que a su vez suponen diferentes tareas. Así, un menú podría ser:

1. Creación de un fichero.
2. Lectura de un fichero.
3. Altas.
4. Bajas.
5. Actualización.
6. Reorganización.
7. Fin.

Al seleccionar, normalmente, por teclado alguna opción se realiza la parte del programa o subrutina que se indique. Mediante la sentencia ON-GOTO sería:

```
100 ON A GOTO 1000, 2000, 5000, 3000...
```

PROGRAMA DE MUESTRA: DEMO 1

```
100 REM DEMO1
120 REM
140 PRINT "MENU"
160 PRINT
180 PRINT
200 PRINT "SELECCIONE OPCION 1, 2, 3"
220 PRINT "1- CREACION DE FICHERO"
240 PRINT "2- MODIFICACION DE FICHERO"
260 PRINT "3- LISTADO DE FICHERO"
280 INPUT A
300 ON A GOTO 500, 600, 1000
320 PRINT "SELECCIONE CORRECTAMENTE

340 GOTO 200
.
.
.
500 PRINT "CREACION DE FICHERO"
.
.
.
.   ...........
580 GOTO 200
600 PRINT "MODIFICACION DE FICHERO"
.
.
.
.   ...........
980 GOTO 200
1000 PRINT "LISTADO DE FICHERO"
.
.
.
.   ...........
2000 GOTO 200
...  END
```

Ejecución

RUN

(menú)

```
SELECCIONE OPCION 1, 2, 3
1- CREACION FICHERO
2- MODIFICACION DE FICHERO
3- LISTADO DE FICHERO
? 5  ______________________________  Se introdujo número
SELECCIONE CORRECTAMENTE             de opción errónea.
SELECCIONE OPCION 1, 2, 3
1- CREACION DE FICHERO
2- MODIFICACION DE FICHERO
3- LISTADO DE FICHERO
? 2
MODIFICACION DE FICHERO ________     Continúa el programa
                                     a partir de aquí hasta
                                     la línea que devuelve de
                                     nuevo el control
                                     al principio.
```

PROGRAMA DE MUESTRA: DEMO 2

Este programa calcula el precio total de un coche, sumando al precio franco de fábrica el impuesto correspondiente según su potencia en CV.

1 a 8 CV –	14%
8 a 12 CV –	20%
12 a 16 CV –	25%
Mayor de 16 CV –	40%

Para su resolución se introduce un menú que permita seleccionar el impuesto correspondiente (porcentaje fijo sobre precio franco de fábrica).

1. 1 a 8 CV (14%)
2. 8 a 12 CV (20%)
3. 12 a 16 CV (25%)
4. Mayor de 16 CV (40%)

```
100 REM DEMO 2
120 REM
140 INPUT "INTRODUCIR PRECIO F.F.";P
160 PRINT "SELECCIONE IMPUESTO"
200 PRINT "1-1 A 8 CV"
220 PRINT "2-8 A 12 CV"
240 PRINT "3-12 A 16 CV"
260 PRINT "4-MAYOR 16 CV"
280 INPUT I
300 ON I GOTO 320, 360, 420, 460
310 PRINT "SELECCIONE CORRECTAMENTE"
315 GOTO 160
320 PRINT "PRECIO TOTAL"; P+P*0.14
340 GOTO 140
360 PRINT "PRECIO TOTAL"; P+P*0.20
400 GOTO 140
420 PRINT "PRECIO TOTAL"; P+P*0.25
440 GOTO 140
460 PRINT "PRECIO TOTAL"; P+P*0.40
480 GOTO 140
600 END
```

```
RUN
INTRODUCIR PRECIO F.F.? 480000
SELECCIONE IMPUESTO
1-1 A 8 CV
2-8 A 12 CV
3-12 A 16 CV
4-MAYOR 16 CV
? 2
PRECIO TOTAL 576000
```

6.1. Sugerencias de programación de ON-GOTO

a) El funcionamiento de ON- GOTO cuando el valor de la variable o expresión es mayor o menor que el número de líneas, aunque en general todos los BASIC siguen la regla dada en el párrafo anterior, le recomendamos lo ensaye y en cualquier forma añada sentencias de recordatorio, reenviando el programa al menú de selección tal como se ha hecho en los programas DEMO 1 y DEMO 2, con las líneas 320-340 y 310- 315, respectivamente.

Después de ON-GOTO introduzca recordatorios para evitar errores al seleccionar una opción del menú.

```
PRINT "SELECCIONE CORRECTAMENTE"
GOTO N
```

b) Al redondear el valor de la variable/expresión de la sentencia ON-GOTO, por ejemplo en:

```
ON A GOTO 400, 500, 600, 200
```

Puede ocurrir que se redondee a un valor ligeramente inferior al previsto, por lo que al truncar la parte entera nos podríamos encontrar con valores no deseados. Así, por ejemplo, si A se redondea por defecto a 2.99 (parte entera 2) en vez de al valor esperado 3.00 (parte entera 3), el control del programa pasaría a la línea 500, con lo que si se deseaba ir a la línea 600 se cometería un error. En aquellos casos que debido al valor de A se pueda prever este caso, será preciso añadir al valor de A una pequeña fracción que evite ese inconveniente. En el caso anterior bastaría sumar 0.1 a la variable A.

```
ON A+0.1  GOTO 400, 500, 600, 200
```

Si A vale 2.99, A+0.1 es 3.09, por lo que su parte entera es 3 y el control del programa pasaría a la línea 600.

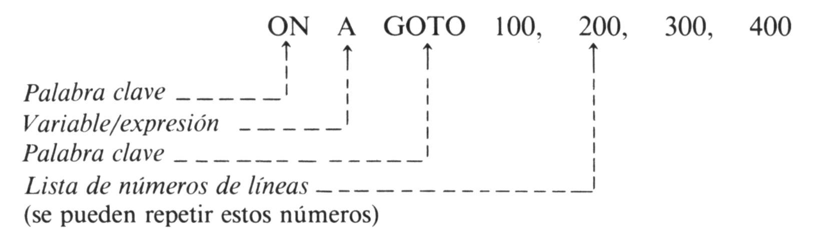
c) Si la expresión detrás de A no es una variable, tenga cuidado al obtener su valor en el programa.

```
ON H-1  GOTO  400, 500, 600
```

Si H−1=1 indica que H=2 y se salta a línea 500
Si H−1=2 indica que H=3 y se salta a línea 600
Si H−1=0 indica que H=1 y se salta a línea 400

6.2. Resumen sentencia ON-GOTO

El formato suele adoptar la forma siguiente:



VARIANTES DE ON-GOTO

La sentencia ON-GOTO, según los intérpretes de BASIC, se utiliza también con otras palabras claves para bifurcar a otra parte del programa cuando ocurre una condición especificada. Las más importantes son:

```
ON-ERROR-GOTO
ON A RESTORE
```

En capítulos posteriores se tratarán estas sentencias.

SUSTITUCION DE ON-GOTO

Si su computadora no tiene la sentencia ON-GOTO, se puede realizar la misma función mediante sentencias IF-THEN múltiples consecutivas.

```
ON A GOTO 400, 500, 600, 700
```

equivale a:

```
IF A=1  THEN 400
IF A=2  THEN 500
IF A=3  THEN 600
IF A=4  THEN 700
```

7. EL BUCLE CONDICIONAL. LAS SENTENCIAS WHILE-WEND y UNTIL-NEXT

Las sentencias FOR-NEXT son ideales para repetir instrucciones BASIC un número especificado de veces. Sin embargo, existen ocasiones en que se desearía repetir un cálculo —esto es, un bucle— mientras que una condición dada sea verdadera; el número de bucles requerido puede variar cada vez que se ejecute el programa. Aunque este bucle puede ser construido mediante combinaciones de sentencias IF-THEN, FOR-NEXT y GOTO de BASIC como BASICA y GW BASIC de MS-DOS, así como ANSI de computadoras VAX y HP poseen las sentencias WHILE-WEND y UNTIL-NEXT que facilitan la programación estructurada. Las versiones modernas posee unas sentencias diseñadas específicamente para ejecutar tales bucles condicionales.

7.1. Bucles WHILE-WEND

El formato general del bucle WHILE-WEND es el siguiente:

```
n1 WHILE expresión de la condición
... sentencias del bucle
n2 WEND
```

n_1 y n_2	Números de línea inicial y final del bucle.
expresión	Cualquier expresión numérica o lógica válida.

La expresión es de igual tipo a la utilizada en la sentencia IF-THEN. La expresión se comprueba primero. Si es verdad, las sentencias del bucle se ejecutan. Si es falsa, el control para la primera sentencia correcta después de la sentencia WEND. La variable que controla el bucle se debe actualizar (incrementar) dentro del bucle.

Supongamos que deseamos escribir un programa que imprime una secuencia de números enteros decrecientes y que se detenga cuando se alcance un determinado valor, por ejemplo, .3. Uno de los diversos métodos que se pueden utilizar es:

```
   10 INPUT "Introduce un numero:";X
┌→ 20 PRINT X
│  30 X=X-1
└─ 40 IF X>=3 THEN 20
   99 END

Introduce un numero:8
 8  7  6  5  4  3
```

El bucle realizado por la sentencia de la línea 40 es condicional, esto es, la ejecución del bucle continua mientras que una determinada condición sea reservada; en este caso la condición es que X sea mayor o igual a 1.

Mediante WHILE-WEND se puede realizar el mismo programa, con una ventaja adicional: su lectura e interpretación es más fácil.

```
10 INPUT "Introduzca un numero:":X
20 WHILE X>=3
30 PRINT X:
40 X=X-1
50 WEND
99 END

RUN
Introduzca un numero:8
 8  7  6  5  4  3
```

Como se ve, el programa hace exactamente la misma cosa que el anterior pero con otra estructura. La sentencia WHILE dice al ordenador «Mientras que (o en tanto que) el valor de X sea mayor o igual a 3, ejecute las sentencias entre WHILE y WEND». La sentencia WEND retorna la ejecución del programa a WHILE; WEND define el «final» (END) del bucle WHILE-WEND.

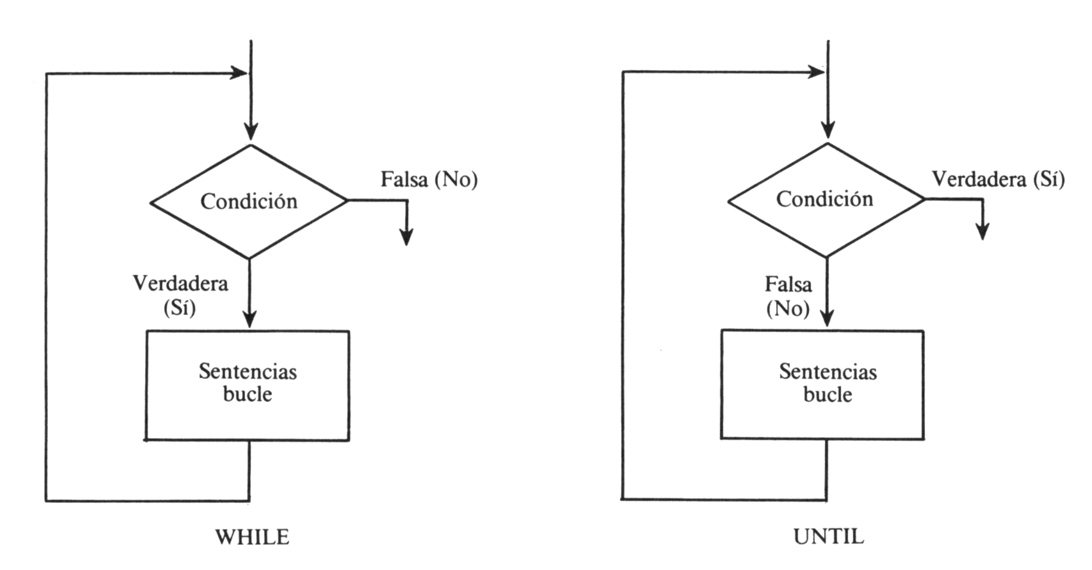
Estará pensando que WHILE-WEND es casi idéntica a FOR-NEXT. Es verdad. Ambos bucles utilizan la misma estructura; las sentencias FOR y WHILE definen el principio del bucle

y las sentencias NEXT y WEND definen el final del bucle. La diferencia, sin embargo, es el método de controlar el bucle: mientras que la sentencia FOR realiza el bucle un número determinado de veces, la sentencia WHILE sólo realiza el bucle mientras que una expresión sea verdadera.

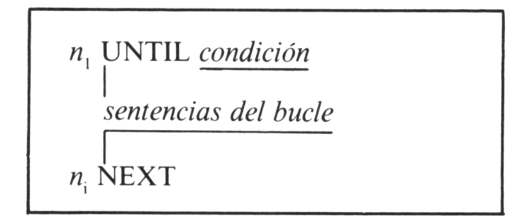
La potencia real de WHILE-WEND se muestra en programas largos y complejos; donde interesa se estructuran los mismos en módulos independientes que faciliten su lectura y comprensión.

7.2. Bucles UNTIL-NEXT

El bucle UNTIL-NEXT es similar al bucle WHILE, pero en este caso la condición se comprueba y se ejecuta el grupo de sentencias completas del bucle cada vez que se encuentra la condición y es falsa. Los diagramas de flujo de WHILE y UNTIL son:



El formato general del bucle UNTIL es:



Los bucles WHILE y UNTIL son más o menos intercambiables. Por ejemplo WHILE X > 0 es lo mismo que UNTIL X < 0. Según los casos puede ser más fácil de utilizar una sentencia u otra.

7.3. Bucles WHILE-WEND anidadas

De igual forma que se pueden anidar bluces FOR-NEXT, se pueden también anidar bucles WHILE- WEND.

```
100 A=1
110 WHILE A<=5
120   B=1
130   WHILE B<=A
140       PRINT "*";
150   WEND
160   A=A+1
170   PRINT
180 WEND
190 END

RUN
*
**
***
****
*****
```

8. SANGRIA EN LOS BUCLES

Es corriente para dar mayor legibilidad a los bucles, escribir las líneas del interior de los mismos, dejando una *sangria* (margen) entre las líneas que fijan el bucle y las líneas interiores.

El siguiente programa calcula los valores de la función y para valores de (x) en el rango −7.5 a +7.5 en incrementos de 0.25.

$y = 5x^3 + 2x^2 - 3$ *ecuación matemática*

programa

```
10 x=-7.5
20 WHILE x<=7.5
30    y=5*x^3+2*x^2-3
40    PRINT "x= ";x;"y= ";y
50    x=x+.25
60 WEND
99 END
```

Este formato se aplica también a sentencias IF-THEN Y FOR-NEXT. Algunos ejemplos se indican a continuación.

1. *Sentencia FOR-NEXT.*

```
10 FOR I=1 TO 5 STEP 1
20     PRINT I
30 NEXT I
```

2. *Sentencia IF-THEN.*

```
10 INPUT A.B
20 IF A=0 THEN GOTO 99
30   MAYOR=A
40   IF B>A THEN MAYOR=B
50   PRINT "EL NUMERO MAYOR ES=": MAYOR
60 GOTO 10
99 END
```

3. *Sentencia IF-THEN-ELSE.*

```
10 IF X<10 THEN
20     P=.10
30 ELSE
40     P=.50
50 'Fin de bucle IF.
```

4. *Sentencias IF-THEN-ELSE anidadas (formato).*

```
10 INPUT X
20 IF X>10 THEN
30   PRINT "Mayor que 10"
40 ELSE
50   IF X<5 THEN
60       PRINT "Menor que 5"
70   ELSE
80       PRINT "Valor 5 a 10"
90   ' Fin segundo bucle
100 ' Fin primer bucle
```

5. *Sentencias WHILE-WEND anidadas.*

```
100 X=1
110 WHILE X<=9
120   Y=1
130   WHILE Y<=9
140     PRINT X;"*";Y;"=";X*Y
150     Y=Y+1
160   WEND
170   X=X+1
180 WEND
190 END
```

PROGRAMAS RESUELTOS

1. *Aplicación de* GOTO *si* N *es* 4, 10 y 25.

```
10 INPUT N
20 PRINT N: PRINT N^2: PRINT N^3
30 GOTO 10
40 END

RUN
? 4
 4
 16
 64
? 10
 10
 100
 1000
? 25
 25
 625
 15625
```

NOTA: *? es la abreviatura de la sentencia* PRINT *en muchas computadoras:* Commodore 64, Amstrad, IBM PC, etc.

2. *Aplicación de* ON-GOTO *si* N *es* 1, 2:

```
10 INPUT N
20 ON  N  GOTO  30,50
30 PRINT "MORTIMER"
40 GOTO 10
50 PRINT "FLANAGAN"
60 GOTO 10
70 END

RUN
? 1
MORTIMER
? 2
FLANAGAN
```

3. *Aplicación de* IF-THEN *si* N *es* 15, 45, 100.

El programa siguiente detecta si el número introducido es mayor de 25.

```
100 REM INTRODUCION DE UN NUMERO
120 INPUT N
140 IF N>25 THEN 220
160 PRINT N; "ES MENOR O IGUAL QUE 25"
180 GOTO 240
```

```
200 REM...MAYOR
220 PRINT N; "ES MAYOR DE 25"
240 GOTO 100

RUN
? 15
 15 ES MENOR O IGUAL QUE 25
? 45
 45 ES MAYOR DE 25
? 100
 100 ES MAYOR DE 25
```

4. *Aplicación de* FOR-NEXT.

Calcular los valores que toma una función algebraica para diferentes valores de la variable X.
Sea la función $Y = x^3 + 5x^2 - 10$. Calcular los valores de *y* para los siguientes valores de *x:*

0 +,6 1,2 1,8 2,4 3 3,6 4,2

```
100 REM CALCULO DE Y
120 FOR X=0 TO 4.2 STEP 0.6
140 Y=X^3+5*X^2-10
160 PRINT X,Y
180 NEXT X
200 END

RUN
 0               -10
 .6              -7.984
 1.2             -1.072
 1.8              12.032
 2.4              32.624
 3                62
 3.6              101.456
 4.2              152.288
```

5. *Aplicación de bucles* FOR- NEXT *anidados.*

El programa siguiente calcula la media de las notas de una clase de *n* alumnos.

```
100 REM PROGRAMA MEDIAS
120 INPUT "NUMERO DE ALUMNOS": N
140 FOR J=1 TO N
160 INPUT "NUMERO DE NOTAS": C
180 S=0
200 FOR I=1 TO C
220 PRINT "NOTA":
240 INPUT A
260 S=S+A
280 NEXT I
300 M=S/C
320 PRINT "MEDIA=": M
340 NEXT J
360 END
```

```
RUN
NUMERO DE ALUMNOS? 2
NUMERO DE NOTAS? 3
NOTA? 4
NOTA? 5
NOTA? 6
MEDIA= 5
NUMERO DE NOTAS? 3
NOTA? 5
NOTA? 6
NOTA? 7
MEDIA= 6
```

6. *Aplicación de bucles* FOR- NEXT *anidados.*

Realización del programa 6 con las siguientes mejoras:

— Introducción del nombre del alumno.
— Introducción del número de orden de las notas.

```
100 INPUT "NUMERO DE ALUMNOS": N
120 FOR I=1 TO N
140 PRINT "NOMBRE DEL ALUMNO": I;
160 INPUT A$
180 PRINT "CUANTAS NOTAS DEL ALUMNO ":A$;
200 INPUT C
220 S=0
240 FOR J=1 TO C
260 PRINT "NORTA NUMERO"; J;
280 INPUT A
300 S=S+A
320 NEXT J
340 M=S/C
360 PRINT
380 PRINT "MEDIA DE ": A$; "=":  M
400 NEXT I
420 END

RUN
NUMERO DE ALUMNOS? 2
NOMBRE DEL ALUMNO 1 ? PEPE
CUANTAS NOTAS DEL ALUMNO PEPE? 2
NORTA NUMERO 1 ? 4
NORTA NUMERO 2 ? 5

MEDIA DE PEPE= 4.5
NOMBRE DEL ALUMNO 2 ? JUAN
CUANTAS NOTAS DEL ALUMNO JUAN? 1
NORTA NUMERO 1 ? 7

MEDIA DE JUAN= 7
```

7. *Aplicación de* ON-GOTO.

Cálculo de la función *y* para un valor de *x* si la mencionada función está definida por:

$y=x^2+5$	*sí*	$x<=0$
$y=3x-1$	*sí*	$0<x<2$
$7=x^2-4x+5$	*sí*	$x>=2$

No se puede utilizar directamente ON- GOTO aunque haya tres posibilidades. Se hacen tres asignaciones a una variable N en función del valor X, de modo que si x <0 N sea 1; si $0<x<2$ N es 2 y si $x>2$ N es 3.

```
100 REM CALCULO DE FUNCION
120 INPUT "VALOR DE X": X
140 IF X<=0 THEN K=1:GOTO 200
160 IF X<2 THEN K=2
180 IF X>=2 THEN K=3
200 ON K GOTO 300, 400, 500
300 Y=X^2+5
310 GOTO 600
400 Y=3*X-1
410 GOTO 600
500 Y=X^2-4*X+5
510 GOTO 600
600 PRINT "Y="; Y
610 GOTO 100

RUN

VALOR DE X? -1
Y= 6
VALOR DE X? 0
Y= 5
VALOR DE X? 1
Y= 2
VALOR DE X? 2
Y= 1
VALOR DE X? 3
Y= 2
```

8. *Aplicación de* FOR-NEXT.

El programa siguiente introduce N calificaciones de una asignatura y calcula su media.

```
100 INPUT "CUANTAS NOTAS": N
120 S=0
140 FOR I=1  TO  N
160 PRINT "INTRODUCIR NOTAS"; I  :
180 INPUT C
200 S=S+C
220 NEXT I
240 M=S/N
260 PRINT "LA NOTA MEDIA ES": M
300 END

RUN

CUANTAS NOTAS? 4
INTRODUCIR NOTAS 1 ? 7
INTRODUCIR NOTAS 2 ? 6.20
INTRODUCIR NOTAS 3 ? 7
INTRODUCIR NOTAS 4 ? 4.80
LA NOTA MEDIA ES 6.25
```

9. *Aplicación de WHILE-WEND.*

El siguiente programa visualiza los cuadrados de los números enteros menores de 300'.

```
10 A=1:B=1
20 WHILE A<300
30   PRINT A;
40   B=B+1
50   A=B^2
60 WEND
99 END

RUN
 1  4  9  16  25  36  49  64  81  100  121  144  169  196
 225  256  289
```

10. *Aplicación de WHILE-WEND.*

El siguiente programa calcula la suma de diferentes números hasta que se introduce la condición final (entrada = 0).

```
10 INPUT N
20  WHILE N<>0
30        INPUT N
40        S=S+N
50 WEND
60 PRINT "SUMA=";S
70 GOTO 10
```

11. *Bucles FOR-NEXT y WHILE-WEND cruzados.*
BASICA del IBM PC y MBASIC, entre otros, permiten cruzar bucles FOR-NEXT y WHILE-WEND del siguiente modo:

```
10 N=3
20 FOR I=1 TO 15
30 M=N+1
40 WHILE M=4
50 N=N+1
60 NEXT I
70 WEND
80 PRINT "M= ";M
90 PRINT "N= ";N
99 END
```

PREGUNTAS Y PROGRAMAS DE REPASO

1. Escribir un programa cuyos resultados de salida sean:

```
1
2
3
4
5
6

FINAL DEL PROGRAMA
```

2. Qué resultado se imprimirá con el siguiente programa:

```
10 FOR A=10  TO  20
20 FOR B=-6  TO  -10  STEP-1
30 PRINT A, B, A*B
40 NEXT B
50 NEXT A
60 END
```

3. Escribir un programa que calcule N, N^2, N^3, para todos los valores de N comprendidos entre 10 y 40 en incrementos de 1. Imprimir los resultados de la computadora.
4. Estudiar las instrucciones y deducir cuáles son las líneas que ejecutará la computadora si se cumplen los valores de las variables que se citan a continuación.

a)

```
200 IF  I>K  THEN  350
     I=18    K=26
```

b)

```
400  IF   M=N  GOTO  60
    M=84    N=70
```

5. Realizar un programa que sume todos los números enteros pares comprendidos entre 2 y 100.
6. Estudiar el siguiente programa:

```
10 A=1
20 B=5
30 PRINT B, B^2
40 A=A+1
50 B=B+5
60 GOTO 30
70 END
```

¿Existe algún error?

¿Cómo se puede solucionar?

7. Escribir un programa que imprima su nombre quince veces, mediante sentencias IF-THEN.

8. Escribir un programa que calcule 1+2+3+4...+25.

9. Escribir un programa que permita sumar los números impares comprendidos entre 1 y 1000.

10. Modificar el programa del problema 9 para que realice la suma de los números pares comprendidos entre 1 y 1000.

11. ¿Cuál es el último valor que toma el índice I en la siguiente instrucción?

```
100 FOR I=-342  TO  1000  STEP  23
```

12. ¿Cuántas veces se imprime la palabra ALELUYA?

```
10 FOR A=1  TO  50  STEP  3
20 PRINT "ALELUYA"
30 NEXT A
40 END
```

13. Escribir un programa que acepte la entrada de un número N, y luego imprima los números mayores que 0 pero menores o iguales a N.

14. ¿Qué imprimirá el siguiente programa cuando se ejecute?

```
10 FOR I=10  TO  1  STEP-1
20 PRINT I; "ABCDEFGHIJ":
30 NEXT I
40 END
```

15. Escribir un programa que dibuje la siguiente figura:

```
*******
 *****
  ***
   *
```

a) Con bucles FOR-NEXT.

b) Con bucles WHILE-WEND.

16. Cuál será la salida del siguiente programa.

```
10 FOR A=1 TO 4
20 FOR B=1 TO 3
30    PRINT A*B
40 NEXT B
50 NEXT A
60 END
```

PROGRAMAS PROPUESTOS

1. Escribir un programa que acepte la entrada de dos números. Si ambos números son mayores o iguales a 10 se imprime la suma. Si ambos números son menores o iguales a 10, imprimir su producto. Si un número es mayor o igual que 10 y el otro es menor que 10, imprimir la diferencia entre el mayor y el más pequeño.
2. Escribir un programa que calcule la media aritmética (suma de N números/N) de N números introducidos por teclado.
3. Realizar un programa que imprima en pantalla el siguiente dibujo.

```
*
**
***
****
*****
******
*******
********
*********
**********
```

4. Escribir un programa que imprima una tabla de cuatro columnas donde se representen los números N, raíz cuadrada de N, raíz cúbica de N y raíz cuarta de N, con el siguiente formato:

```
NUMERO          RAIZ CUADRADA         RAIZ CUBICA          RAIZ CUARTA
1               1                     1                    1
2               1.4142...             1.25992              ...
......          ......                ......               ...
```

NOTA: N se debe introducir por teclado en cada ejecución.

5. Realizar un programa que permita calcular el factorial de cualquier número que se introduce por teclado. Recuerde que FACTORIAL de N (N) = N × (N− 1) × (N− 2) × ...3 × 2 × 1.

   Ejemplo: FACTORIAL de 5=5 × 4 × 3 × 2 × 1=120.

CAPITULO 5

Cadenas de caracteres-funciones de cadena

0. INTRODUCCION

En esta lección introducimos al lector en un tratamiento profundo sobre las variables alfanuméricas o de cadena, así como en las cadenas de caracteres.

En los capítulos 1 y 2 se definieron las cadenas de caracteres y las variables de cadena. En la presente lección se profundizará sobre el tema de modo que al final de la misma el lector podrá:

— Tratar cualquier tipo de variable de cadena.

— Deducir la longitud de caracteres de una cadena.

— Convertir variables numéricas en variables de cadena y viceversa.

— Operar con las diferentes funciones de cadena.

— En general proporcionarán mayor potencia a sus programas en lenguaje BASIC.

1. CADENAS

Ya se ha visto cómo las computadoras mediante las variables y funciones numéricas permiten hacer cálculos sencillos y complejos. Sin embargo, estas variables no permiten el diálogo interactivo computadoras/usuario mediante mensajes y palabras diversas, o la presentación de datos, es decir, el tratamiento de datos no numéricos.

Mediante un grupo de *funciones* llamadas de *cadena* y de *las variables de cadena* ya definidas se podrán tratar datos alfanuméricos, o sea, caracteres numéricos, alfabéticos y símbolos especiales incluidos en los teclados de los terminales de la computadora o, más exactamente, en sus códigos ASCII.

Recordemos el significado de una *cadena* («string» en inglés).

Ejemplos de cadenas:

(a) `"HOLA MARIA"`
(b) `"12 OCTUBRE DE 1982"`
(c) `"JAEN"`

Cualquier colección de letras, símbolos o números, incluyendo palabras reconocibles y espacios, representan *una cadena si están encerrados entre comillas.*

Escriba las cadenas *a, b, c* anteponiéndoles la sentencia PRINT. Al pulsar RETURN habrá observado que se presentan en pantalla los siguientes mensajes:

```
HOLA MARIA
12 DE OCTUBRE DE 1982
JAEN
```

Una variable *alfanumérica* o de *cadena* se representa con el mismo código que las variables numéricas, pero añadiéndoles un signo $.

Variables alfanuméricas válidas son:

```
A$, A1$, B5$, JULIO$, VIC$
```

Casi todos los caracteres del teclado pueden formar parte de una connotación de cadena, con la excepción de las *comillas*(«), debido a que ellas forman parte de la cadena.

Algunos ejemplos son:

```
A$="HOLA"
B$="MARIA"
C$="1234"
D$="30 DE NOVIEMBRE DE 1982"
```

Los espacios en blanco dento de las cadenas se consideran como caracteres y son tratados de igual forma que los restantes símbolos y caracteres.

La *longitud* de las variables de cadenas es el número de caracteres que contiene entre comillas.

"HOLA"	*Longitud 4 caracteres*
12 DE OCTUBRE	*Longitud 13 caracteres*

Las cadenas pueden tener longitud cero, llamándose en este caso *cadena vacía;* se representan por " ".

A$=""	*Cadena vacía o nula*

Las cadenas se pueden definir mediante las sentencias LET, INPUT, READ, DATA y PRINT. El tratamiento de cadenas facilita la *interacción* con la computadora mediante ls sentencias anteriores y las que estudiaremos a lo largo de esta lección, al permitir al usuario el diálogo en «modo directo» con la máquina.

SUMA O CONCATENACION DE CADENAS

Las cadenas se pueden sumar juntas para hacer unas cadenas más largas utilizando un signo más (+) (algunas computadoras utilizan el símbolo &).

```
A$="COMPU"
B$="TADORA"
A$+B$="COMPUTADORA"
C$=A$+B$
```

Atención. Esta línea no es válida a efectos de programación. (referida a A$+B$="COMPUTADORA")

Al sumar las dos cadenas no se ha dejado espacio entre ambas; si lo desea, existen dos caminos para hacerlo.

a)

```
10 A$="ARTILLERIA"
20 B$=" "
30 C$="DE"
40 D$=" "
50 E$="CAMPAÑA"
60 F$=A$+B$+C$+D$+E$
70 PRINT F$

RUN
ARTILLERIA DE CAMPAÑA
```

b)

```
10 A$="ARTILLERIA"
20 C$="DE"
30 E$="CAMPAÑA"
40 F$=A$+" "+C$+" "+E$
50 PRINT F$

RUN
ARTILLERIA DE CAMPAÑA
```

Las variables de cadena se pueden reasignar al igual que las variables numéricas.

```
A$="VIVA"
A$=A$+"ZAPATA"
```

A$ toma el valor VIVAZAPATA.

COMPARACION DE CADENAS

Las cadenas se pueden tratar mediante operadores lógicos al igual que las variables numéricas. Así, los operadores >, <, = funcionan también en las cadenas y de acuerdo a las reglas correspondientes al orden lexicográfico usual; es decir, JUAN será menor que JUANETE.

Para ello se recurre al código ASCII (véase APENDICE I) utilizado por casi todas las computadoras, menos las fabricadas por IBM (con la excepción del IBM/PC), que siguen al código EBCDIC. En el código ASCII de la computadora APPLE II (Applesoft), la letra A se representa por el número decimal 65, mientras que la letra B es el número 66, o sea, que las letras A a la Z se representan por los números desde el 65 al 90.

Mediante la sentencia IF ... THEN y los operadores de relación lógicos y aritméticos podremos realizar comparaciones diferentes.

Así, por ejemplo:

```
10 A$="A"
20 B$="B"
30 IF A$<B$  THEN  100
 .
 .
 .
 .
100  ....
 .
 .
 .
 .
```

En la línea 10 se almacena el carácter A (valor decimal en código ASCII estándar 65) en la variable A$, lo que será igual a tener almacenado el número 65. La línea 20 almacena en la variable B$ el carácter B (66 decimal) o lo que es igual el número 66. La línea 30, en

consecuencia, al comparar A$ y B$, en realidad lo que está comparando es 65 y 66 o lo que es igual 65 < 66. Por consiguiente en este caso el programa continuaría en la línea 100.

De acuerdo con lo anterior, IF X$ > Y$ THEN... comprobaría si el valor que representa X$ es mayor que el valor que representa Y$.

De igual modo se podría realizar con los operadores >= (mayor o igual) y <= (menor o igual).

```
IF A$>=B$ THEN...
```

Ejemplos:

a) `A$<B$`

equivale a preguntar si la cadena A$ está clasificada en el código ASCII antes que la cadena B$. Si A$="A" y B$="H", A$ < B$.

b) `A$="A"` `B$="ABC"` *Implica que A$ < B$*

Todos los caracteres de una cadena se comparan uno a uno, con los de la otra cadena, hasta encontrar la primera diferencia.

c)
```
10 A$="CASA"
20 B$="ARBOL"
30 IF A$>B$ THEN 100
 .
 .
 .
 .
100 ....
 .
 .
```

En este caso, se comparan en primer lugar las letras iniciales del texto C y A. C es mayor que A, por tanto CASA es mayor que ARBOL y el programa al llegar a la línea 30 saltaría a la línea 100.

d)
```
10 A$="GARAGE"
20 B$="GARCIA"
30 IF A$>B$ THEN 100
40 ...
 .
 .
 .
100
```

Las tres primeras letras, en este caso GAR, son iguales y la cuarta letra A es menor que C; en consecuencia, A$ es menor que B$ y en este caso el programa pasaría de la línea 30 a la línea 40.

e) Además de letras y números, se pueden comparar también símbolos o caracteres especiales.

```
10 A$="/"
20 B$="#"
30 IF A$>B$ THEN 100
 .
 .
 .
100 PRINT "OK"
120 END
```

El símbolo (/) tiene por código 47, y el símbolo (#), 35.

2. FUNCIONES DE CADENA

Como ya se ha mencionado, una de las aplicaciones más útiles de las computadoras es el proceso y tratamiento de datos no numéricos. Las funciones de cadena son una «herramienta» muy potente para dicho tratamiento.

El número de funciones de cadena disponibles varía de una computadora a otra. En la tabla 1 se indican las utilizadas por las computadoras más populares.

Tabla 5.1. Funciones de cadena de diferentes computadoras.

Commodore 64/128	ASC, CHR$, VAL, LEFT4, LEN, MID$, RIGHT$, STR$, GET$.
APPLE II	ASC, CHR$, LEFT$, MID$, RIGHT$, STR$, VAL, LEN, GET$.
DRAGON 32/64/200	ASC, CHR$, HEX$, INKEY$, INSTR, LEFT$, LEN, MID$, RIGHT$, STRING-$, STR$, VAL.
HP85/HP87	STRING$, LEN, POS, VAL, NUM, CHR$, UPC$.
ATARI	ASC, CHR$, LEN, STR$, VAL.
ZX Spectrum	CHR$, INKEY$, LEN, SCREEN$, STR$, VAL, CODE, n_1 TO n_2
IBM/PC	ASC, CHR$, DATE$, HEX$, INKEY$, INSTR, LEN, MID$, MKI$, STR$, SPACE$, STRING$, LEFT$, RIGHT$, MID$, MKI$, MKS$, MKD$, OCT$, VARPTR$, INPUT$.
MSX	ASC, CHR$, BIN$, HEX$, INSTR$, LEF$, LEN, MID$, RIGHT$, STR$, STRING$, VAL, OCT$, SPACE$, INPUT .
Amstrad	ASC, CHR$, HEX$, INSTR, VIN$, UPPER$, LOWER$, LEFT$, RIGHT$, MID$, LEN, STR$, VAL, SOPACE$, STR, NG$.

2.1. LEN

A lo largo de un programa suele necesitarse conocer el número de caracteres de una cadena o de varias. Para ello se dispone de la función LEN.

La función LEN cuenta el número de caracteres de una cadena. La palabra LEN es la asbreviatura de LENGHT (longitud).

Se cuentan los caracteres no imprimibles y los espacios en blanco.

```
10 A$="VIVA ZAPATA"
20 N=LEN (A$)
30 PRINT N

RUN
 11
```

Longitud de la cadena A$

Formato:

n LEN (A$)

n Número de línea.
A$ Variable de cadena.

La variable de cadena, cuya longitud se requiere, es el argumento de la función.

APLICACIONES

```
A$="MARIA"
N=LEN (A$)
```

equivale a N = 5

b) `PRINT LEN (A$)`
c) `IF LEN (A$)=8  THEN 100`
d) `M=3*LEN (A$)-1` *equivale a M = 14*
e) `K=LEN ("18 DE MAYO DE 1982")` *equivale a K=18*
f)
```
10 A$="MURCIE"
20 B$="LAGO"
30 N=LEN (A$+B$)
A$+B$="MURCIELAGO"
```
N toma el valor 10

Ejemplo:

```
10 A$=""
20 B$="   "
30 C$="0123456789"
40 PRINT LEN (A$); LEN (B$); LEN (C$)
50 PRINT LEN (A$+B$+C$)

RUN
 0  3  10
 13
```

El resultado es evidente, ya que:

A$	*Es la cadena nula o vacía y contiene 0 caracteres.*
B$	*Tiene 3 caracteres.*
C$	*Tiene 10 caracteres.*
A$+B$+C$	*Tiene 0+3+10=13 caracteres.*

Entre las aplicaciones prácticas de la función LEN se pueden destacar:

1) La edición de textos, por ejemplo, para subrayar una palabra cuya longitud se desconoce de antemano.
2) El proceso letra a letra de una palabra y su análisis cuando no se conoce previamente.

2.2. VAL

Una cadena puede estar compuesta de letras, números o caracteres especiales y en ocasiones puede ocurrir que una variable numérica o de cadena representen un mismo número y ser, sin embargo, distintas.

A=47	*A Es una variable numérica.*
A$="47"	*A$ Es una variable de cadena.*

Esta diferencia es muy importante y de hecho en algunas partes de los programas se suelen utilizar con mucha frecuencia todas las variables como de cadena, con lo que se evita el problema de confundir datos numéricos por cadenas y viceversa, y cuando se necesita se realiza el proceso de conversión de una cadena o variable alfanumérica en una variable numérica mediante la función VAL.

Veamos una aplicación para justificar su empleo:

```
100 A=20
120 B=40
140 C=A+B
160 PRINT C
```

El resultado de la suma A+B es 60. Pero supongamos que el programa anterior se realiza con variables de cadena.

```
100 A$="20"
120 B$="40"
140 C$=A$+B$
160 PRINT C$
```

El resultado en este caso es 2040, ya que las cadenas se han sumado.

En resumen, el tratamiento de datos suele ser muy cómodo hacerlo mediante cadenas, ya que de este modo se pueden introducir números o caracteres alfabéticos o especiales. Sin embargo, cuando estos datos son números, se precisará convertir estas variables de cadena a numéricas para poder operar con números. Esta misión es la que realiza la función VAL.

Formato: *n* VAL *(cadena o variable de cadena)*

n	Número de línea.
cadena, variable de cadena	A$, "BUENOS DIAS" "1234",...

La función VAL permite la conversión de una cadena o variable de cadena en una variable numérica.

REGLAS PRACTICAS

1. Si la cadena/variable de cadena comienzan con letras, gráficos o caracteres no numéricos, el empleo de la función VAL produce un 0.

```
A$="MORGAN"        VAL (A$)=0
B$="MORTIMER"      VAL (B$)=0
C$="MCKOY"         VAL (C$)=0
```

Este proceso se puede comprobar en un programa.

```
100 PRINT  VAL  ("MORGAN")
120 A$="HP41"
140 PRINT VAL (A$)

RUN
 0
 0
```

Como resultado se obtienen dos ceros.

2. Si la cadena está compuesta exclusivamente de números, la función VAL convierte esta cadena en el número correspondiente.

```
A$="121"           VAL (A$)=121
B$="1234"          VAL (B$)=1234
C$="456"           VAL (C$)=456
```

Comprobemos de nuevo este proceso en un programa.

```
10 N=VAL ("121")
20 A$="4321"
30 PRINT N, VAL (A$)

RUN
 121            4321
```

3. Si la cadena comienza con números, pero contiene también letras o símbolos, la función VAL convierte esta cadena en valor numérico inicial con independencia de las letras y de cualquier número que le siga.

```
G$="12AB"        VAL (G$)=12
H$="4XYZ"        VAL (H$)=4
I$="925H"        VAL (I$)=925
```

APLICACIONES

Esta función puede ser útil en programas en que se verifique si una entrada es o no adecuada. Veamos un caso típico.

```
100 INPUT "ESCRIBA SU NOMBRE": N$
140 PRINT N$
```

Al ejecutar el programa se puede contestar en la línea 100 al signo de interrogación correspondiente, con palabras o números, por ejemplo, “MORTIMER” o “485”. Es decir, si la variable de entrada es de cadena se pueden introducir palabras y números.

Añada al programa anterior la línea 120.

```
100 INPUT "ESCRIBA SU NOMBRE": N$
120 IF VAL (N$)<>0 THEN 100
140 PRINT N$
```

Ejecútelo y responda con un número.

```
RUN
ESCRIBA SU NOMBRE? 58
ESCRIBA SU NOMBRE? PEPE
PEPE
```

El programa volverá siempre a la línea 100 mientras VAL (A$) sea distinto de cero. VAL (A$) es sólo igual a cero cuando A$ no es un número, esto es, si A$ es una letra o combinación de caracteres.

Existe una teoría entre programadores experimentados que aconseja utilizar en todas las sentencias de entrada variables de cadenas tales como A$ y que las variables numéricas tales como A se deben evitar en lo posible. El motivo es que si se necesita una entrada numérica basta con utilizar todo el conjunto de funciones de cadena para tratar y formatear los datos, de modo que sería difícil, si nó imposible, hacerlo con números.

En resumen, mediante variables de cadena se podrán leer o introducir números, o caracteres indiferentemente.

Otra aplicación muy usual de VAL es en el tratamiento de ficheros, ya que en estos casos se precisa con gran frecuencia mezclar caracteres alfanuméricos y leerlos todos en forma de carácter. Por ejemplo, si una ficha contiene nombre, dirección, profesión, teléfono y distrito postal, es mucho más práctico y flexible hacer que todos los campos sean variables de cadenas, con lo cual el tratamiento será similar para todas.

2.3. LEFT$

Con la ayuda de las siguientes funciones, LEFT$, RIGHT$ y MID$, será posible extraer cualquier subcadena (parte de una cadena) de cualquier otra cadena. Todas ellas tienen en común que precisan el símbolo $ al final de la palabra clave. Comencemos con la función LEFT$.

La función LEFT$ es una función de cadena que permite extraer de una cadena un cierto número de caracteres comenzando por la izquierda. Dicho de otra forma, LEFT$ devuelve una cadena compuesta de los *N* caracteres más a la izquierda que figura como argumento.

Sentencia	*Impresión*
100 PRINT LEFT$ ("MCKOY",1)	M
120 PRINT LEFT$ ("MCKOY",2)	MC
200 A$="12345"	
220 PRINT LEFT$ (A$,4)	1234

Formato:

n LEFT$ (A$,N)

n Número de línea.
A$ Cadena o variable de cadena.
N Número de caracteres a extraer a partir de la izquierda.

Si la longitud a tomar es mayor (>) que la longitud de la cadena, se toma la cadena completa. Si la longitud a tomar es 0, se devuelve una cadena vacía o nula.

EJEMPLOS DE APLICACION

a)

```
10 A$="INSTITUTO DE ENSENANZA MEDIA"
20 B$=LEFT$ (A$,9)
30 PRINT B$
40 C$=LEFT$ (A$,22)
50 PRINT C$
60 D$=LEFT$ (A$,28)
70 PRINT D$
80 END

RUN
INSTITUTO
INSTITUTO DE ENSENANZA
INSTITUTO DE ENSENANZA MEDIA
```

b) La función LEFT$ es muy útil en programas para comprobar la entrada examinando sólo su primera letra.

```
100 PRINT "ELIJA OPCION"
120 PRINT "COMENZAR DE NUEVO"
```

```
140 PRINT "LECTURA DE LISTADO"
160 PRINT "GUARDAR EN DISCO"
200 INPUT A$
220 IF LEFT$ (A$,1)="C" THEN 1000
240 IF LEFT$ (A$,1)="L" THEN 2000
260 IF LEFT$ (A$,1)="G" THEN 3000
```

En este pequeño programa se comprueba la primera letra de la respuesta y se compara con la primera de cada respuesta posible. Si coinciden, se salta a la línea 1000, 2000 o 3000 según la mencionada letra.

c) En el siguiente programa vemos cómo se pueden introducir cadenas por teclado y luego visualizar un formato con todas las posibles combinaciones.

```
120 INPUT "INTRODUZCA UNA FRASE": A$
140 FOR N=1  TO  LEN (A$)
160 PRINT LEFT$ (A$,N)
180 NEXTN
```

Al ejecutar el programa supongamos se introduce «VIVA ZAPATA».

```
RUN
INTRODUZCA UNA FRASE? VIVA ZAPATA
V
VI
VIV
VIVA
VIVA
VIVA Z
VIVA ZA
VIVA ZAP
VIVA ZAPA
VIVA ZAPAT
VIVA ZAPATA
```

d) Pequeña subrutina que permite detectar dos posibles respuestas: Sí (S), No (N).

```
100 INPUT "INTRODUZCA LA RESPUESTA"; A$
120 IF LEFT$ (A$,1)="S" THEN 1000
140 IF LEFT$ (A$,1)="N" THEN 2000
160 PRINT "PRUEBE DE NUEVO"
180 GOTO 100
```

Mediante esta subrutina, si las respuestas no comienzan por S o N, la línea 180 retorna a la línea 100 hasta que el usuario introduce una de las dos respuestas correctas S o N.

2.4. RIGHT$

Esta función es muy similar a LEFT$. Permite extraer de una cadena un cierto número de caracteres a la derecha, o lo que es lo mismo devuelve los N caracteres de la derecha de una cadena.

La diferencia entre LEFT$ y RIGHT$ es evidente. Con LEFT$ se puede seleccionar una parte de una cadena comenzando por la izquierda, mientras que con RIGHT$ se puede seleccionar una parte de una cadena comenzando por la derecha.

Sentencia	*Impresión*
100 PRINT RIGHT$ ("MARIA",1)	A
120 PRINT RIGHT$ ("MARIA",2)	IA
140 PRINT RIGHT$ ("MARIA",5)	MARIA
200 FOR N=1 TO 5	A
220 PRINT RIGHT$ ("MARIA",N)	IA
240 NEXTN	RIA
	ARIA
	MARIA

Ejemplos

a)

```
10 A$="INSTITUTO DE ENSENANZA MEDIA"
20 B$=RIGHT$ (A$,5)
30 PRINT B$
40 C$=RIGHT$ (A$,15)
50 PRINT C$
60 D$=RIGHT$ (A$,28)
70 PRINT D$
80 END

RUN
MEDIA
ENSENANZA MEDIA
INSTITUTO DE ENSENANZA MEDIA
```

b)

```
10 A$="VIVA ZAPATA"
20 FOR I=1  TO  LEN (A$)
30 PRINT RIGHT$ (A$, I)
40 NEXT I
50 END

RUN
A
TA
ATA
PATA
APATA
ZAPATA
 ZAPATA
A ZAPATA
VA ZAPATA
IVA ZAPATA
VIVA ZAPATA
```

c) El siguiente programa ilustra la diferencia esencial entre la función LEFT$ y RIGHT$.

```
100 A$="VIVA ZAPATA"
120 FOR N=1  TO  LEN (A$)
140 PRINT LEFT$ (A$, N), RIGHT$ (A$, N)
160 NEXTN

RUN
V                A
VI               TA
VIV              ATA
VIVA             PATA
VIVA             APATA
```

```
VIVA Z          ZAPATA
VIVA ZA          ZAPATA
VIVA ZAP        A ZAPATA
VIVA ZAPA       VA ZAPATA
VIVA ZAPAT      IVA ZAPATA
VIVA ZAPATA     VIVA ZAPATA
```

2.5. MID$

Esta es probablemente la más útil de las funciones de formateado de cadena y es incluso la más sofisticada.

Formato:

n MID$ (A$, I, J)

n Número de línea.
A$ Variable de cadena o expresión de cadena.
I Principio de la posición.
J Longitud a considerar o tomar.

Tanto I como J deben estar comprendidos entre 0 y 244, obteniendo error en caso contrario. Si no hay longitud a tomar o existen menos caracteres a la derecha del principio de la posición que los pedidos por la longitud a tomar, esta función restituye todos los caracteres a la derecha de la posición inicial.

Si el principio de la posición es mayor que la longitud de la cadena, la función devuelve una cadena nula o vacía.

Sentencia	*Impresión*
10 PRINT MID$ ("MORGAN", 3, 2)	RG
20 PRINT MID$ ("MORGAN", 3, 4)	RGAN
30 PRINT MID$ ("MORGAN", 4)	GAN

En las sentencias anteriores se ve que si J no existe se presentan todos los caracteres a partir del carácter I.

```
100 A$="ABCDE"
120 FOR I=1  TO  4
140 PRINT MID$(A$, 2, I)
160 NEXT I

RUN
B
BC
BCD
BCDE
```

En resumen, una sentencia MID$ (A$, 4, 6) significa construir una cadena de seis caracteres de longitud que comienza en el carácter cuatro de la cadena A$.

Ejemplos:

a)
```
10 A$="HOLA FLANAGAN"
20 FOR N=1  TO  LEN (A$)
30 PRINT MID$ (A$, N)
40 NEXT N
```

```
RUN
HOLA FLANAGAN
OLA FLANAGAN
LA FLANAGAN
A FLANAGAN
 FLANAGAN
FLANAGAN
LANAGAN
ANAGAN
NAGAN
AGAN
GAN
AN
N
```

b)

```
10 A$="HOLA FLANAGAN"
20 FOR N=1  TO  LEN (A$)
30 PRINT MID$ (A$, N, 1), MID$ (A$, N, 2)
40 NEXT N

RUN
H               HO
O               OL
L               LA
A               A
                 F
F               FL
L               LA
A               AN
N               NA
A               AG
G               GA
A               AN
N               N
```

PROGRAMAS DE APLICACION DE LEFT$, RIGHT$, MID$

1) Programa con las tres funciones incorporadas.

```
10 A$="LUIS GARCIA LOPEZ"
20 N$=LEFT$ (A$, 4)
30 A1$=RIGHT$ (A$, 5)
40 A2$=MID$ (A$, 6, 6)
50 PRINT N$, A1$, A2$
60 END

RUN
LUIS                LOPEZ               GARCIA
```

2) Sustitución de unos caracteres de una variable de cadena por otros distintos.

```
10 A$="SR. GARCIA JOSE LUIS"
20 MID$ (A$, 5, 6)="XXXXXX"
30 PRINT A$

RUN
SR. XXXXXX JOSE LUIS
```

3)

```
10 A$="1234567890"
20 MID$ (A$, 3, 4)="XXXXXX" :REM MODIFICACION 3456
30 PRINT A$
40 MID$ (A$, 3, 4)="3456"                :REM MODIFICACION 3456
50 MID$ (A$, 3, 4)="XX"     :REM MODIFICACION 34
60 PRINT A$
70 END

RUN
12XXXX7890
12XX567890
```

Esta posibilidad de sustitución no existe más que para la instrucción MID$. No se puede realizar con las funciones LEFT$ y RIGHT$. Solamente la función MID$ puede figurar a la izquierda del signo =.

4) El programa siguiente acepta una entrada de un número del 1 al 7, el cual representa los días de la semana, es decir, el lunes es 1, el martes es 2, etc. Al ejecutarse el programa se convierte dicho número en las tres primeras letras del día correspondiente.

```
100 INPUT "INTRODUCIR NUMERO DEL DIA": D
120 D$="..LUNMARMIEJUEVIESABDOM"
140 A$=MID$ (D$, 3*D, 3)
160 PRINT A$

RUN
INTRODUCIR NUMERO DEL DIA? 2
MAR
```

En la línea 140, la función MID$ se utiliza para seleccionar las tres letras correspondientes de la cadena A$. Así nuestro ejemplo como D=2 A$=MID$ (D$, 6, 3), por consiguiente se extraen tres caracteres a partir del sexto carácter.

Como ejercicio para el lector, realice la rutina anterior para el caso de los doce meses del año.

2.6. ASC (cadena)

Esta función convierte la primera letra o carácter de una cadena en su valor numérico correspondiente en código ASCII.

Formato:

n ASC *(cadena o variable de cadena)*

n	Número de línea.
cadena, variable de cadena	Válida.

```
10 A$="TEST"
20 PRINT ASC (A$)
30 END

RUN
 84                         El código del carácter T es 84
```

Si cambia la línea 10 por:

```
10 A$="BUENOS DIAS"
```

Al ejecutar el programa:

```
RUN
 66
```

El código ASCII del carácter B es 66

Ejemplos:

```
100 A=ASC ("A")
120 B=ASC ("ALMERIA")
140 A$="ALICANTE" : C=ASC (A$)
160 PRINT A; B: C

RUN
 65  65  65
```

En este caso se ha impreso el número 65 para las tres funciones.

b) Un método para obtener los códigos ASCII asociados a cada carácter de una cadena y no sólo el primero es la siguiente rutina.

```
100 A$="PET"
120 FOR I=1  TO  LEN (A$)
140 B$=MID$ (A$, I, 1)
160 A=ASC (B$)
180 PRINT B$; A
200 NEXT I
220 END

RUN
P 80
E 69
T 84
```

2.7. STR$

Esta función es la opuesta de VAL. Convierte un número en una cadena de caracteres.

Formato: | *n* STR$ (X) |

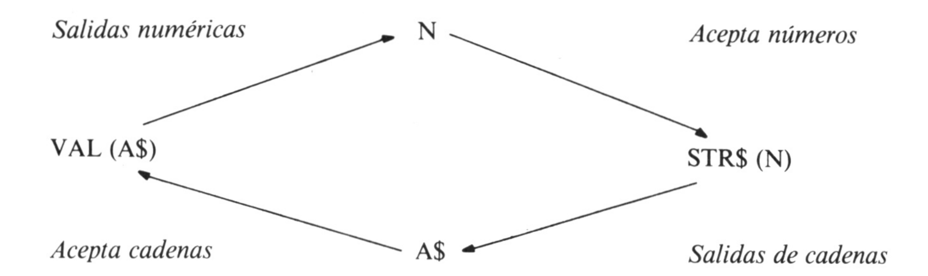
n Número de línea.
X Número a convertir.

El primer carácter de la cadena se reserva para el signo, que será un espacio para los números positivos y un signo «–» para los números negativos.

```
10 A=123
20 A$=STR$ (A)
30 PRINT A, A$
40 END

RUN
 123    123
```

Como se ve, tanto A como A$ imprimen en su resultado 123, sin embargo son variables diferentes.



PROGRAMAS DE APLICACION

a) Examine el siguiente programa y trate de ver sus resultados.

```
10 A=44.5
20 B$=STR$ (A)
30 FOR N=1  TO  LEN (B$)
40 C$=MID$ (B$, N, 1)
50 C=VAL (C$)
100 PRINT C
120 NEXT N
140 END
```

En la línea 10, A representa una variable numérica. La línea 20 convierte dicha variable en una cadena. Las líneas siguientes aplican las técnicas de formateado de cadenas a B$, con lo que se obtienen los diferentes caracteres aislados de la misma.

¿Ha averiguado los resultados que obtendrá? En caso contrario ejecute el programa en su computadora.

b) El programa siguiente muestra una concatenación mediante funciones STR$.

```
10 INPUT X, Y
20 PRINT "X="; X, "Y="; Y
30 Z=X+Y
40 PRINT "SU SUMA ES"; Z
50 X$=STR$ (X)
60 Y$=STR$ (Y)
70 PRINT "X$="; X$, "Y$="; Y$
80 Z$=X$+Y$
90 PRINT "LA CONCATENACION ES"; Z$

RUN
? 425, 327
X= 425              Y= 327
SU SUMA ES 752
X$= 425             Y$= 327
LA CONCATENACION ES 425 327
```

Como se ve, la función STR$ deja un espacio en blanco del signo +. Si se efectúa una segunda ejecución con valores negativos se tendrá:

```
RUN
? 180, -380
X= 180              Y=-380
SU SUMA ES-200
X$= 180             Y$=-380
LA CONCATENACION ES 180-380
```

2.8. CHR$

Es la función inversa de ASC. Permite generar caracteres alfanuméricos correspondientes a un código dado (valor entre 0 y 255) que depende del valor del argumento.

Formato:

n CHR$ (X)

n Número de línea.
X Valor comprendido entre 0 y 255.

Así, PRINT CHR$ (66) visualiza B, ya que el código ASCII de la letra B es 66.
PRINT CHR$ (72) visualiza H (código ASCII de H, 72).
En el Apéndice I se indica la relación entre los caracteres ASCII y sus números de código correspondientes de diferentes computadoras.
Así, por ejemplo, ciñéndonos a los códigos especiales, se puede ver que aparte de imprimir caracteres puede realizar determinadas actividades.

PRINT CHR$(7)	*Activa un timbre interno de la computadora.*
PRINT CHR$(10)	*Se produce un salto de línea o avance de línea.*
PRINT CHR$(13)	*Provoca un retorno de carro.*
PRINT CHR$(42)	*Imprime un asterisco (*).*

PROGRAMAS DE APLICACION

1. En ocasiones, bien porque el fabricante no lo proporciona en su *Manual de Instrucciones* (es muy raro que esto suceda), o por cualquier otra causa, se precisa disponer del código ASCII completo de nuestra computadora, ya que la utilidad del mismo como estamos viendo a lo largo del capítulo es muy grande.

La siguiente subrutina imprimirá el código completo. Al ejecutar la subrutina en la primera columna se visualiza el número decimal entre 0 y 255 y en la segunda columna sus caracteres equivalentes.

```
10 FOR I=1  TO  255
20 PRINT I. CHR$ (I)
30 NEXT I
40 END
```

2. Listado parcial código ASCII.

```
100 INPUT "NUMERO INICIAL Y FINAL": A, B
120 FOR I=A  TO  B
140 PRINT I; CHR$ (I)
160 NEXT I
180 END
```

Ejecutamos este programa:

```
RUN
NUMERO INICIAL Y FINAL? 65, 90
 65 A
 66 B
 67 C
 68 D
 69 E
```

```
70 F
71 G
72 H
73 I
74 J
75 K
76 L
77 M
78 N
79 O
80 P
81 Q
82 R
83 S
84 T
85 U
86 V
87 W
88 X
89 Y
90 Z
```

NORMAS PRACTICAS DE PROGRAMACION

1. Esta función se utiliza con mucha frecuencia en el diálogo entre unidad central y terminales, para enviar caracteres especiales a los mismos, principalmente impresora y pantalla.

2. Mediante la función CHR$ se pueden realizar en modo programa, impresiones y representaciones gráficas que sólo en modo directo son posibles.
El código ASCII del símbolo comillas (») es 34.
Trate de escribir en la pantalla de la computadora mediante una sentencia PRINT el siguiente mensaje:

```
MI NOMBRE ES "MORGAN"
```

Observará que no es fácil la empresa, ya que si escribe

```
PRINT "MI NOMBRE ES MORGAN"
```

o una frase similar no lo conseguirá.

Sin embargo, escriba la siguiente instrucción:

```
PRINT "MI NOMBRE ES ";CHR$(34); "MORGAN"; CHR$(34)
```

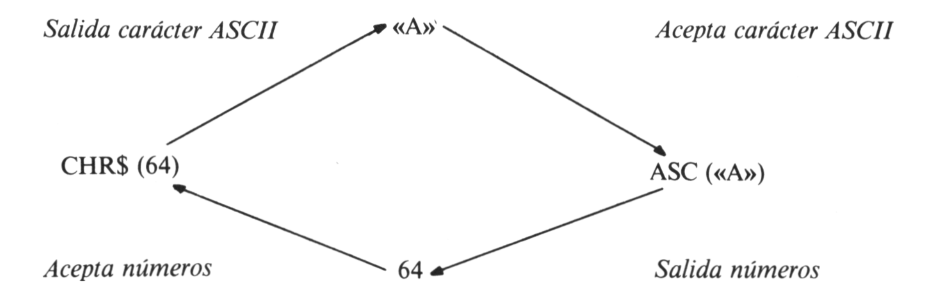
Al ejecutar esta línea obtendrá el mensaje buscado:

```
MI NOMBRE ES "MORGAN"
```

De igual modo se podrían utilizar los códigos ASCII para *borrado de pantalla, video inverso, movimientos de cursor,* etc.

3. Mediante la función CHR$ se pueden realizar comparaciones y clasificaciones de cadenas de caracteres, dado que nos permite comparar los códigos ASCII correspondiente de los caracteres y ver la relación de *comparación entre* los mismos.

Comparación CHR$-ASC



Las funciones de cadena analizadas hasta este momento se consideran básicas y suelen residir en casi todas las versiones de BASIC y se resumen en la tabla 5.2.

Tabla 5.2. Funciones de cadena básicas.

Función	*Utilizada en*	*Proporciona*
LEN (...)	cadenas	números
LEFT$ (...)	cadenas	cadenas
RIGHT$ (...)	cadenas	cadenas
MID$ (...)	cadenas	cadenas
VAL (...)	cadenas	número
STR$ (...)	números	cadenas
ASC (...)	caracteres	número
CHR$ (...)	cadenas	carácter

3. FUNCIONES DE CADENA ESPECIALES

Existen otras diferentes funciones de cadena que si no tan importantes como las anteriores, suelen ya venir incorporadas en diversas versiones de BASIC como son MICROSOFT, CBASIC, etc.

Se describen a continuación la más usuales:

3.1. SPACE$ (X)

Esta función permite generar una cadena de caracteres o espacios en blanco cuya longitud es X.

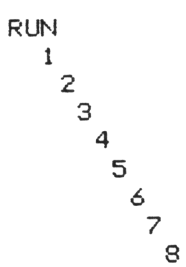
La expresión X se redondea a un entero y debe tener un valor comprendido entre 0 y 255.

Formato:

n SPACES (X)

n Número de línea.
X Expresión numérica.

```
10 I=1
20 L$=SPACE$ (I)
30 PRINT L$; I
40 I=I+1
50 IF I=<8 THEN GOTO 20
60 END
```



3.2. STRING$

La función STRING$ devuelve una cadena cuya longitud se especifica por medio de la expresión entera. Los caracteres de la cadena obtenida tienen el mismo código ASCII que se haya especificado para la segunda expresión entera o que comience con el primer carácter de la expresión de cadena especificada.

Formato: | *n* STRING$ *(n_1, n_2 $)* |

n Número de línea.
n_1 Expresión entera que indica la longitud.
n_2 Expresión entera o cadena (dos formatos: ASCII, teclado).

Ejemplos:

```
A$=STRING$ (75, "A")
B$=STRING$ (60. 25)
```
STRING$ con caracteres de teclado.
STRING$ con caractres código ASCII.

PROGRAMAS DE APLICACION

1)
```
10 A$="INFORME MENSUAL"
20 B$=STRING$ (10, 42)
30 PRINT B$; A$; B$
40 END
```

n_1 en la función STRING$ es 10, lo que significa que la longitud del carácter representado por n_2 $, en este caso «42» («*» en código ASCII), será de 10.
En consecuencia, al ejecutarse el programa se tiene:

2)
```
RUN
**********INFORME MENSUAL**********

10 A$=STRING$ (5, "*")
20 PRINT A$
30 END

RUN
*****
```
*Genera una cadena de 5 **

3) La siguiente subrutina permite dar la misma longitud a diferentes cadenas de caracteres. Así, por ejemplo, veamos cómo se podrían imprimir diferentes nombres de personas con igual

longitud rellenando con puntos los espacios en blanco que le falten a cada uno hasta el total máximo prefijado.

```
100 INPUT "INTRODUCIR NOMBRE"; N$
120 PRINT N$; STRING$ (25-LEN (N$), ".")
130 INPUT "SI DESEA CONTINUAR ESCRIBA SI"; R$
140 IF R$="SI" THEN 100
200 END
```

```
LUIS GARCIA GARCIA.......
RAMON CID................
JOSE LUIS PEREZ..........
ANTONIO RAMIREZ..........
```

Resultados sucesivos de la ejecución de la línea 120, con los nombres especificados.

4) Impresión de caracteres específicos.

```
PRINT STRING$ (10, "-")
----------
PRINT STRING$ (25, "X")
XXXXXXXXXXXXXXXXXXXXXXXXX
```

3.3. INSTR

Esta función permite obtener la posición de una cadena de caracteres dentro de otra cadena (encontrar la posición o la cadena que se busca por primera vez a partir de la posición inicial).

Formato: n INSTR *(m, cadena principal, cadena buscada)*

n	Número de línea.
m	Posición inicial (si no existe *m* se toma 0).
cadena principal	Cadena o variable de referencia.
cadena buscada	Cadena o variable a localizar.
m	Está comprendido entre 0 y 255.

La función INSTR devuelve un 0 en cualquiera de los siguientes casos:

a) Si la posición inicial *m* es > longitud de la cadena principal.
b) Si la cadena principal está vacía.
c) Si la cadena buscada no está contenida es la cadena principal.

La función INSTR devuelve la posición inicial o 1 si la cadena buscada es nula o vacía.

Ejemplos de aplicación:

a)
```
10 A$="SR. MORTIMER. MCKOY"
20 A=INSTR (5. A$. ".")
30 B$=LEFT$ (A$. A-1)
40 PRINT B$
50 END

RUN
SR. MORTIMER
```

En efecto, A contiene la posición del punto situado entre MORTIMER y MACKOY, ya que se pide la búsqueda a partir del carácter M, que es el carácter 5.

b)
```
PRINT INSTR (3, "MNMMNMM", "N")
 5
```

3.4. Entrada del teclado sin pulsación de las teclas ENTER o RETURN

La sentencia INPUT tratada en el capítulo 3 es el método más adecuado para permitir al usuario la introducción de información en un programa que se está ejecutando. Sin embargo, existen otros dos métodos de introducción de información que pueden ser más apropiados para ciertas apliaciones. Estas son las funciones GET, INPUT$ e INKEY$. Una diferencia importante entre INPUT y estas dos nuevas funciones es que mientras INPUT requiere del usuario la pulsación de la tecla ENTER para indicar a la computadora que lea la información escrita, INPUT$, INKEY$ y GET no lo necesitan.

3.4.1. GET

Aunque está sentencia no es una función de cadena, sí tiene su mayor utilidad en el tratamiento de cadenas.

Formato:

n GET *variable numérica* *n* GET *variable de cadena*

La sentencia GET acepta un carácter numérico de teclado y GET C$ un carácter alfanumérico.

Mediante la sentencia GET se evita la pulsación de la tecla RETURN en numerosas operaciones.

El empleo más utilizado es GET C$, ya que de este modo la aceptación de un carácter puede ser el representado por cualquier tecla.

Su forma más usual es:

```
100 GET A$: IF A$="" THEN 100
```

La pareja de comillas (" ") significa código nulo. Al llegar el programa a la línea 100 se detiene la ejecución en la línea 100 hasta que se pulsa cualquier tecla (código distinto de «nulo») en cuyo cso se pasa el control a la siguiente línea de programa. Mientras no se pulse ninguna tecla se permanece en un bucle sin fin en la línea 100.

Así pues, una forma usual de trabajar con esta sentencia es en las interrupciones voluntarias de programación al objeto de permitirle realizar operaciones externas, consultas de datos, etc.

```
100 PRINT "CUANDO ESTE PREPARADO PULSE CUALQUIER TECLA"
120 GET A$: IF A$="" THEN 120
130 PRINT A$
140 END
```

Este pequeño pograma espera en la línea 120 a que se pulse cualquier tecla, en cuyo instante se ejecutará la línea 130 presentando en pantalla precisamente ese carácter.

GET se puede programar para esperar la introducción de una tecla específica.

```
10 GET A$
20 IF A$<>"F" THEN 10
30 PRINT A$
40 END
```

Hasta tanto no se pulsa la tecla F, el programa estará bloqueado en las líneas 10-20.
El siguiente ejemplo muestra cómo GET se puede utilizar para controlar programas:

```
10 PRINT "COMIENZA EL JUEGO"
  .
  .
  .
100 PRINT "DESEA JUGAR S/N"
200 GET A$
210 IF A$="" THEN 200
220 IF A$="S" THEN 10
230 PRINT "FIN DEL JUEGO"
```

El bucle 200-210 se realiza en él hasta que se pulsa una tecla que permite continuar el programa en la línea 220. Si se pulsa la tecla «S», se transfiere el control a la línea 10; si se pulsa cualquier otra tecla, «N» o distinta, se ejecuta la línea 230.

La ventaja de este programa es que se puede introducir una entrada (S o N) sin más que pulsar una tecla (S o N) y sin pulsar RETURN.

3.4.2. INKEY$

La sentencia o función INKEY$ es equivalente a GET C$ en el MICROSOFT BASIC, por ejemplo en las computadoras IBM PC, MSX, Amstrad, etc.

Formato: | *n variable de cadena*=INKEY$ |

Cuando el programa llega a esta línea, se explora el teclado para ver si ha pulsado alguna tecla, en cuyo caso se almacena en la variable de cadena especificada.

No requiere la pulsación de la tecla RETURN.

Ejemplos:

```
400 A$=INKEY$: IF A$="" THEN 400
```

Se espera la pulsación de cualquier tecla

```
100 A$=INKEY$
120 IF A$="" THEN PRINT "FIN"
130 PRINT "OK"
```

Al pulsar una tecla se imprimirá la palabra OK y en el caso contrario «FIN».

3.4.3. INPUT$ (INPUT *sin* ENTER)

INPUT$ permite introducir caracteres desde el teclado sin necesidad de pulsar ENTER:

Formato: | *n variable de cadena* = INPUT$ *(expresión numérica)* |

INPUT es una función de cadena que devuelve una cadena de caracteres del teclado sin requerir la pulsación de ENTER. El número de caracteres devuelto es el valor de la expresión

numérica del argumento INPUT$. Aceptan todos los caracteres del teclado (incluyendo caracteres de control) excepto CONTROL BREAK.

El siguiente ejemplo muestra como funciona INPUT$.

```
10 PRINT "Pulse tres teclas cualesquiera:"
20 A$=INPUT$(3)
30 PRINT "Su entrada ha sido "A$
99 END
```

Al ejecutar este programa, en pantalla aparece:

```
RUN
Pulse dos teclas cualesquiera:
```

La pantalla se ha quedado «congelada». No hay signo de interrogación, típico de INPUT, ni cursor parpadeante. El programa, simplemente, está esperando la pulsación de tres teclas.

Pulse las teclas f, i, n. Al instante de pulsar la tecla n, el programa responderá con:

```
Su entrada ha sido fin
Ok
```

Observe que no tiene necesidad de pulsar ENTER para que el programa lea y procese su entrada. De igual modo notará que las letras f, i y n no aparecen inmediatamente en la pantalla, como sucedía con la sentencia INPUT. INPUT$ no produce em de la información de entrada (impresión de los caracteres pulsados en pantalla), como hace INPUT.

El valor de argumento de INPUT puede estar comprendido entre 1 y 255.

La cadena de caracteres pulsados se ha asignado a la variable de cadena A$. Mientras que INPUT se puede utilizar sola en una línea, INPUT$ no puede. INPUT$ es una función que devuelve un valor de cadena desde el teclado. Debe aparecer junto con una sentencia BASIC que procese el valor de la función. Por ejemplo, se puede imprimir su valor con la sentencia PRINT INPUT$ (3).

Aplicación

INPUT$ se puede aplicar con gran eficacia en la selección de opciones de los diferentes menús de un programa, consideremos el programa DEMO1 de muestra del capítulo 4. La línea 280 es una sentencia INPUT típica.

```
280 INPUT A
```

Esta sentencia requiere del usuario, que tras la selección de la opción se pulse ENTER para completar la entrada. (La información precedente, líneas 200-260, da información sobre el menú y las instrucciones para introducir un número del 1 al 3.) Si desea escribir este menú, de tal modo que el usuario haga la selección pulsando sólo una de las teclas 1 a 3 sin necesidad de pulsar posteriormente ENTER, sustituya la línea 280 por las dos siguientes líneas:

```
280 OP$=INPUT$(1)
290 OPCION=VAL(OP$)
```

La nueva línea 230 asigna la entrada del teclado a la variable OP$. La línea 290 convierte el valor de la cadena OP$ en un valor numérico que pueda ser utilizado posteriormente en la sentencia ON-GOTO.

SINTESIS DE LAS FUNCIONES DE CADENA

Instrucción	*Manipulación de texto*
A$=LEFT$ (B$, N)	Entrega una subcadena A$ que contiene los N primeros caracteres de B$.
A$=RIGHT$ (B$, N)	Da una subcadena A$ que contiene los N últimos caracteres de B$.
A$=MID$ (B$, M, N)	Entrega una subcadena A$ extraída de B$ a partir del M° carácter y conteniendo N caracteres.
A=LEN (B$)	Entrega la longitud de caracteres de la cadena B$.
A=INSTR (N, A$, B$)	Entrega la posición de la cadena B$ en la cadena A$ a partir del N° carácter (obtiene un 0 si B$ no está en A$).
C$=CHR$ (N)	Entrega un carácter del código ASCII correspondiente al número decimal N.
A=VAL (A$)	Convierte la variable de cadena A$ en una variable numérica. A$ debe tener como primer carácter un dígito precedido de un signo (+) o (−); en caso contrario, A valdrá 0.
A$=STR$ (N)	Convierte la variable numérica N en una variable de cadena.
A$=SPACE$ (N)	Entrega una cadena que contiene N espacios en blanco.
A$=STRING$ (N, N$)	Entrega una cadena A$ que contiene N caracteres idénticos al «carácter» o al código numérico correspondiente expresado en N$.
GETA$/GETA	Acepta un carácter alfanumérico o numérico del teclado.
A$=INKEY$	Acepta caracteres alfanuméricos del teclado.
A$=INPUT$(N)	Acepta una cadena de N caracteres del teclado.

PROGRAMAS RESUELTOS

PROGRAMA 1 (*Aplicación de* LEFT$):

```
10 A1$="BUENOS DIAS SEVILLA"
20 A2$=LEFT$ (A1$, 7)
30 PRINT A1$: LEN (A1$)
40 PRINT A2$: LEN (A2$)

RUN
BUENOS DIAS SEVILLA 19
BUENOS  7
```

```
10 A$="MORTIMER"
20 N=LEN (A$)
30 FOR I=1 TO N
40 PRINT LEFT$ (A$, I)
50 NEXT I
60 END

RUN
M
MO
MOR
MORT
MORTI
MORTIM
MORTIME
MORTIMER
```

PROGRAMA 2 (*Aplicación* RIGHT$):

```
10 A$="BUENOS DIAS"
20 N=LEN (A$)
30 FOR I=1 TO N
40 PRINT RIGHT$ (A$.I)
50 NEXT I

RUN
S
AS
IAS
DIAS
 DIAS
S DIAS
OS DIAS
NOS DÍAS
ENOS DIAS
UENOS DIAS
BUENOS DIAS
```

PROGRAMA 3 (*Aplicación* de MID$):

```
10 A$="MORTIMER"
20 FOR I=1 TO 5
30 B$=MID$ (A$, 3, I)
40 PRINT B$
50 NEXT I
60 END

RUN
R
RT
RTI
RTIM
RTIME
```

— *Escritura a la inversa de una cadena de caracteres.*

```
10 INPUT N$
20 L=LEN (N$)
30 FOR I=L TO 1 STEP-1
40 N1$=MID$ (N$, I .1)
50 PRINT N1$;
60 NEXT I

RUN
? MORTIMER
REMITROM
```

PROGRAMA 4 (*Aplicación de concatenación de cadenas*):

```
10 A1$="MORTIMER"
20 A2$="FLANAGAN"
30 A$=A1$+" "+A2$
40 PRINT A$

RUN
MORTIMER FLANAGAN
```

— Inversión de una cadena de caracteres.

```
10 INPUT A$
20 L=LEN (A$)
30 I$=""
40 FOR I=L TO 1 STEP-1
50 A1$=MID$ (A$, I , 1)
60 I$=I$+A1$
70 NEXT I
80 PRINT I$
90 END

RUN
? MORTIMER
REMITROM

RUN
? MCKOY
YOKCM
```

PROGRAMA 5 (*Aplicación de* STRING$):

```
10 FOR I=1  TO  8
20 PRINT STRING$ (I, "*")
30 NEXT I
40 END

RUN

*
**
***
****
*****
******
*******
********
```

```
10 FOR I=1 TO 8
20 A$=STRING$ (I, " ")
30 B$=STRING$ (I, "*")
40 D$=A$+B$
50 PRINT D$
60 NEXT I

RUN

 *
  **
   ***
    ****
     *****
      ******
       *******
        ********
```

PROGRAMA 6 (*Aplicación de* VAL):

```
10 INPUT N$
20 P=VAL (N$)
30 PRINT P

RUN

? 34.237
 34.237
```

PROGRAMA 7 (*Aplicación de* STR$):

```
10 INPUT N
20 A$=STR$ (N)
30 L=LEN (A$)
40 PRINT "*"; A$: "*": L

RUN

? 8
* 8* 2
```

PROGRAMA 8 (*Aplicación de* CHR$):

Programa que escribe las 26 letras del alfabeto.

```
10 FOR I=65 TO 90
20 PRINT CHR$ (I):
30 NEXT I
40 END


RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

PROGRAMA 9 (*Aplicación de* ASC):

```
10 INPUT A$
20 FOR I=1 TO LEN (A$)
30 Z$=MID$ (A$, I, 1)
40 PRINT Z$, ASC (Z$)
50 NEXT I
60 END

RUN
? PEPITO
P                   80
E                   69
P                   80
I                   73
T                   84
O                   79
```

PROGRAMA 10 (*Aplicación de* INPUT$):

```
100 PRINT "Pulse S continuar/P parar"
110 A$=INPUT$(1)
120 IF A$="S" THEN 400
130 IF A$="P" THEN END
.
.
400 . . .
```

Este programa lee un solo carácter procedente del teclado en respuesta a la pregunta.

PROGRAMA 11 (*Aplicación de* INPUT$):

Este programa introduce al concepto de «palabra de pago» o «palabra de acceso» cuyo concepto se asemeja al de clave, ya que tiene por objeto permitir que el programa o determinadas partes del mismo, sólo pueda ser ejecutado por el usuario autorizado.

El siguiente subprograma sólo permite el paso al programa principal, cuando el usuario introduce una palabra clave autorizada (en nuestro caso situadas en una sentencia DATA). La palabra clave debe tener 8 letras y ser alguna de las tres palabras autorizadas de la sentencia DATA en la línea 40.

```
10 REM ***palabra clave***
20 REM
```

```
30 REM
40 DATA "Mackoy;:","Mcfarlan","Camberio",""
50 REM
60 PRINT "Introduzca su palabra clave"
70 PALCLAVE$=INPUT$(8)
80 PRINT
90 RESTORE
100 READ CODIGO$
110 WHILE CODIGO$ <> PALCLAVE$ AND CODIGO$ <> ""
120 READ CODIGO$
130 WEND
140 IF CODIGO$="" THEN PRINT "Palabra clave falsa":GOTO 50
150 PRINT "Palabra de acceso aceptada"
160 REM
170 REM ***Programa principal***
180 ...
```

La palabra de acceso no aparece en la pantalla, lo que proporciona más seguridad al programa. Como afirmación de este concepto, si usted es usuario de alguna tarjeta de crédito, recuerde que durante las operaciones de retirada de fondos en un cajero automático, se visualizan todas las cantidades que se introducen por teclado excepto su número o código clave.

PREGUNTAS Y PROGRAMAS DE REPASO

1. ¿Cómo se construye una variable que represente caracteres alfanuméricos y cómo se denomina?
2. ¿Cuántos caracteres puede tener una cadena?
3. ¿Las variables de cadena pueden ser completamente numéricas?; es decir, ¿esta instrucción es legal?

```
100 A$="12345"
```

4. ¿Se pueden comparar las variables de cadena? ¿Es legal esta instrucción?

```
100 IF A$<>B$ THEN 100
```

5. ¿Se pueden utilizar valores alfanuméricos para cálculos? ¿Cómo podría conseguirlo?
6. ¿Es correcta la sentencia 100 A$=«12 DE OCTUBRE ***»?
7. ¿Qué imprimirá el siguiente programa?

```
10 A$="ESTO ES"
20 B$="UNA PRUEBA"
30 PRINT A$; B$
40 PRINT B$; A$
50 END
```

8. Escribir un programa que imprima las tres palabras ROMA, WELLINGTON, BARCELONA, en orden alfabético.
9. ¿Qué misión realiza la función LEN?
10. ¿Qué imprimirá este programa?

```
10 INPUT A$
20 PRINT LEN (A$)
```

Su respuesta al ejecutarse la sentencia INPUT es:

a) `PENTECOSTES`
b) `MORTIMER`
c) `" "`

11. ¿Para qué sirven las funciones LEFT$, RIGHT$, MID$?

12. ¿Qué resultados se imprimirán en el siguiente programa?

```
10 INPUT N$
20 FOR I=1 TO LEN (N$)
30 PRINT LEFT$ (N$, I); RIGHT$ (N$, I)
40 NEXT I
```

Su respuesta al ejecutarse la sentencia INPUT es:

a) ROMA
b) MORTIMER

13. ¿Qué se imprimirá al ejecutarse el siguiente programa?

```
10 A$="MARGARITA"
20 A1$=LEFT$ (A$, 0)
30 A2$=MID$ (A$,4 ,7)
40 PRINT A1$; A2$
```

14. ¿Qué se entiende por concatenar?

15. ¿Cuál es la diferencia entre

`ASC ("H")` y `("HOLA")`

16. ¿Qué se obtendrá al ejecutar las siguientes sentencias?

```
PRINT CHR$ (ASC ("B"))
PRINT CHR$ (ASC ("BARCELONA"))
```

PROGRAMAS PROPUESTOS

1. Deducir un programa que permita determinar el número de dígitos de un número entero que se introducirá por teclado.

2. Escribir un programa que permita escribir un número entero cualquiera en diez caracteres, sustituyendo los espacios en blanco a la izquierda por el símbolo «*».

   Ejemplo:

Número introducido		*Impresión*
128	*se escribirá*	*******128
43215	*se escribirá*	*****43215

3. Escribir un programa que permita a justar o encuadrar (justificar) por la derecha una serie de números introducidos mediante teclado.

   Ejemplo: Números introducidos 3.21, 4, 15.21, 743.2, 841, 1.

   El resultado de la impresión deberá ser:

```
  3.21
  4.00
 15.21
743.20
841.00
  1.00
```

4. Escribir un programa que permita cifrar un texto que se introduce mediante teclado, siguiendo el siguiente código: la letra A se sustituye B, la letra B por C, la letra C por D..., la letra Z se sustituirá por la letra A.

   Ejemplo: Si se introduce MARIA, se habrá de imprimir y cifrar NBSJB.

5. Realizar un programa que permita determinar el número de cifras de un número entero introducido por teclado mediante INPUT.

6. Se desea realizar un programa de modo que cualquier número entero (máximo siete cifras) se represente con asteriscos a la izquierda cuando tenga menos de siete cifras, es decir:

723	*se representaría*	****723
56143	*se representaría*	**56143

7. Realizar un programa de criptografía de textos que realice la siguiente función: el texto que se introduce mediante INPUT se debe cifrar de modo que se obtenga un texto cifrado tal que cada letra del mensaje original se sustituya por la siguiente del alfabeto.

Ejemplo:

```
ACADEMIADEARTILLERIA    (texto normal)
BDBEFNJBEFBSUJMMFSJB    (texto en clave o cifrado)
```

8. Realizar un programa que permita deducir el día de la semana introduciendo su número de orden, de tal modo que si se introduce 1, se lea LUNES; 2, MARTES, etc., y en caso de introducir 0, 8, 9, etc. se obtenga error.

9. Realizar un programa que calcule el nombre del mes cuando se introduzca el número correspondiente en el año.

```
1-ENERO
2-FEBRERO
----------
```

10. Realizar un programa que permita calcular los días transcurridos desde el 1 de enero al día que se desee del año.

CAPITULO 6

Funciones

0. INTRODUCCION

Todas las versiones de BASIC proporcionan un cierto número de subrutinas incorporadas que no se conocen con el nombre de *funciones* y que le proporcionan una flexibilidad y una potencia al lenguaje que sin ellas sería muy difícil de conseguir en algunas ocasiones, especialmente en cálculos matemáticos, y que de obtenerlo exigiría normalmente un aumento de la memoria central de la computadora.

Las funciones se dividen en tres grandes grupos:

a) Definidas en el propio BASIC o incorporadas al mismo.
b) Definidas por el usuario.
c) Funciones especiales.

Al final de este capítulo el lector podrá:

— Conocer e interpretar todas las funciones estándar incluidas en el intérprete BASIC de la computadora como:

SQR, ABS, INT, EXP, LOG, SONG, ...

— Definir sus propias funciones utilizando las funciones estándar o bien los operadores aritméticos y lógicos del sistema.
— Utilizar funciones/sentencias tales como PEEK, POKE, FRE, MEM, etc.

1. FUNCIONES ESTANDAR O INCORPORADAS AL SISTEMA

De igual modo que las calculadoras portátiles, programables o no, todas las versiones de BASIC disponen de funciones numéricas o estándar que se encuentran incorporadas al sistema y que pueden definirse fácilmente mediante los formatos adecuados.

Las funciones numéricas se aplican sobre un argumento que puede ser una constante, variable o expresión aritmética.

Las funciones numéricas son en realidad matemáticas.

El juego de funciones estándar no es igual en todas las computadoras, por lo que siguiendo con los criterios establecidos al principio del libro, recogeremos las más importantes y que la mayoría de las versiones BASIC admiten. Le aconsejamos que consulte la lista de funciones que soporta su computadora antes de trabajar con ellas.

Las funciones se dividen en cuatro grupos:

— Generales.
— Logarítmicas y exponenciales.
— Trigonométricas.
— Especiales.

2. FUNCIONES NUMERICAS GENERALES

Son aquellas funciones que permiten la realización de cálculos aritméticos tales como: valores absolutos, enteros, máximos, mínimos, raíces cuadradas... de números.

La más importantes son:

ABS, FIX, INT, MAX, MIN, RND, SGN, SQR.

2.1. Función ABS

La función ABS devuelve el valor absoluto del argumento ABS(X)=X para X mayor o igual que 0 y ABS(X)=−X para X menor que ∅. En aritmética el valor absoluto de X se representa por | X |.

Formato: ABS (X)

Ejemplos:

Y=ABS(−1) es equivalente a Y=Valor absoluto de (−1)=1.
Z=ABS(3.45) es equivalente a Z=Valor absoluto de (3.45)=3.45.

```
10 X=-5.23: A=ABS (X)
20 Y= 4.25: B=ABS (Y)
30 PRINT A: B

RUN
 5.23  4.25
```

2.2. Función FIX

Devuelve una función truncada del argumento. Todos los dígitos a la derecha del punto decimal se truncan o eliminan, resultando un valor entero.

Formato: FIX (X)

Para X no negativo. FIX(X) es igual a INT(X)
Para X negativo. FIX(X) es igual a INT(X)+1

Ejemplo 1:

```
FIX (2.2)=2
FIX (-2.5)=-2
Y=ABS (A-FIX (A))
```

Esta última sentencia da a Y la parte decimal o fraccionaria de A.

Ejemplo 2:

```
100 A=3.41425
120 B=ABS (A-FIX (A))
140 PRINT A; B

RUN
 3.41425  0.41425
```

2.3. Función INT

INT es la abreviatura de INTEGER (entero).

Devuelve la parte entera del argumento y, por consiguiente, nos permitirá obtener la parte entera de cualquier cantidad.

Recordemos que la parte entera de un número es el entero inmediatamente inferior a este número.



Ejemplos:

```
INT (-4.6)=-5
INT (-4.2)=-5
INT (-2.5)=-3
INT ( 2.5)= 2
INT ( 4.3)= 4
```

Así, para el caso de números X negativos las expresiones:

`ABS (INT (X))`   `INT (ABS (X))`   *no son iguales*

```
PRINT ABS(INT(-1.31)); INT(ABS(-1.31))
 2  1
```

NOTA: *La diferencia entre* FIX *e* INT *aparece sólo con números negativos.* FIX *trunca, simplemente, todos los dígitos después del punto decimal.* INT *devuelve el entero mayor que sea menor o igual al argumento.*

EJEMPLOS DE UTILIZACION

a) Redondeo de números

En numerosas operaciones matemáticas se requiere redondear un número cualquiera al entero más próximo, de modo tal que si la última cifra es 5 o mayor que 5, se redondea *por exceso,* es decir, al entero inmediatamente superior; si la última cifra es menor que 5, se *redondea por defecto,* es decir, al entero correspondiente.

4.5 *se debe redondear a* 5
4.3 *se debe redondear a* 4

El medio más adecuado para conseguir el mencionado redondeo es añadir 0.5 al número considerado antes de tomar la parte entera.

`INT (4.3+0.5)=INT (4.8)=4`   *El número 4.3 se redondea a 4*
`INT (4.8+0.5)=INT (5.3)=5`   *El número 4.8 se redondea a 5*

Ejemplo: El siguiente programa obtiene la media aritmética de dos números, suma de ambos dividida por 2 y la redondea a su parte entera.

```
10 INPUT A, B
20 M=(A+B)/2
30 M=INT (M+0.5)
40 PRINT M

RUN
? 4, 5
 5

RUN
? 6, 7.4
 7

RUN
? 3, 5.3
 4
```

b) Redondeo a n *cifras decimales*

Normalmente, en la mayoría de los cálculos matemáticos se requieren aproximaciones o redondeos a una o varias cifras decimales. Supongamos que se desea redondear cualquier número real de varias cifras decimales a sólo dos, décimas y centésimas. El procedimiento consistiría en multiplicar el número considerado por 100, sumarle 0.5, obtener la parte entera de dicha suma y dividir el resultado por 100.

`7.345627` *redondeado con 2 cifras decimales es* `7.35`. *La codificación sería:*

```
INT (7.345627*100+0.5)/100=7.35
```

Ejemplo:

```
100 REM REDONDEO DE NUMEROS
120 INPUT A
140 B=INT (A*100+0.5)/100
160 PRINT B

RUN
? 13.456789
 13.46

RUN
? 18.432432
 18.43
```

Si se desea redondear a *n* cifras decimales, es preciso multiplicar por una potencia de 10 elevada a *n*, sumarle 0.5 y hallar la parte entera del resultado.

Ejemplo: Redondear todos los números introducidos por teclado hasta la cuarta cifra decimal.

```
100 INPUT A
120 B=INT (A*10^4+0.5)/10^4
140 PRINT B
```

```
RUN
? 7.435678
 7.4357

RUN
? 4.3214161
 4.3214
```

c) Detección de números enteros

Con frecuencia se puede desear conocer si un número introducido por teclado, calculado a lo largo de un proceso o leído en sentencias DATA (se estudiarán en el capítulo 7), es entero o no.

El procedimiento es muy sencillo y basta comparar el número con su parte mediante una sentencias IF-THEN. Si son iguales, el número será entero.

```
100 INPUT "INTRODUCIR EL NUMERO"; N
120 IF N=INT (N) THEN 200
140 PRINT N; "NO ES ENTERO"
160 STOP
200 PRINT N; "ES ENTERO"

RUN
INTRODUCIR EL NUMERO? 6
 6 ES ENTERO

RUN
INTRODUCIR EL NUMERO? 4.25
 4.25 NO ES ENTERO
```

2.4. Funciones MAX y MIN

La función MAX se utiliza para determinar cuál es el valor máximo de una serie de expresiones numéricas.

Formato:

a) MAX (*expresión numérica*[1], *expresión numérica*$_2$, ...)
b) Y=*expresión numérica*$_1$ MAX *expresión numérica*$_2$

Ejemplos:

`MAX (A, B, C, D)`	*entrega el valor máximo de A, B, C, D*
`Y=A MAX B`	*entrega el valor máximo de A y B*

La función MIN se utiliza para determinar cuál es el valor mínimo de una serie de expresiones numéricas.

Formato:

a) MIN (*expresión numérica*$_1$, *expresión numérica*$_2$, ...)
b) Y=*expresión numérica*$_1$ MIN *expresión numérica*$_2$

Ejemplo:

```
100 INPUT "CUATRO VALORES": A, B, C, D
120 PRINT "CUATRO VALORES"; A; B; C; D
140 PRINT "MAXIMO": MAX (A, B, C, D)
160 PRINT "MINIMO": MIN (A, B, C, D)

RUN
CUATRO VALORES ? 1, 7, 4, 9
CUATRO VALORES 1  7  4  9
MAXIMO 9
MINIMO 1
```

Las computadoras Amstrad y HP soportan estas funciones en sus versiones BASIC.

2.5. Función CINT

CINT es la abreviatura de «Convert to Integer» (convertir a entero).

Convierte el argumento en un número entero.

Formato: CINT(X)

Si X no está en el rango −32768 a 32767, se producirá un error «Overflow» (desbordamiento). X es un número de simple o doble precisión y CINT lo convierte a un valor entero, mediante la típica operación aritmética de *redondeo* al entero más próximo (6.4 se redondea a 6, 6.8 a 7 y 6.5 a 7).

Obsérvese la diferencia entre las tres funciones de conversión a enteros.

```
PRINT CINT(55.67)      PRINT INT(55.67)      PRINT FIX(55.67)
 56                     55                    55

PRINT CINT(-2.78)      PRINT INT(-2.78)      PRINT FIX(-2.78)
-3                     -3                    -2
```

La figura 6.1 muestra gráficamente el funcionamiento y las diferencias entre las tres funciones de conversión:

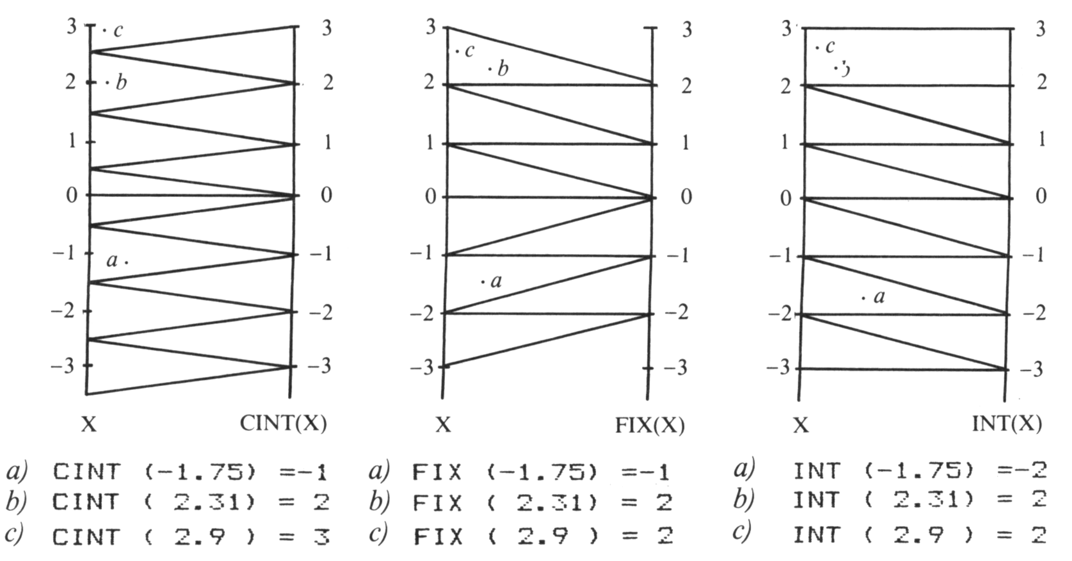


Figura 6.1. Conversión de enteros con CINT, FIX e INT.

FIX e INT «truncan» el operando al valor de la parte entera, mientras que CINT redondea al entero más próximo. Otra diferencia entre CINT y las dos funciones FIX e INT es que mientras CINT devuelve un entero verdadero, tanto FIX como INT devuelven números de simple precisión que tienen una parte decimal de 0.

2.6. Cambio de precisión en números

A veces es importante poder cambiar convertir un número de doble precisión a un número de simple precisión o viceversa. Por ejemplo, estas conversiones pueden servir para ahorrar memoria (conversión de doble a simple precisión) o representar un número con 14 dígitos (doble precisión) a un formato de 7 dígitos (simple precisión). Las funciones que realizan estas tareas son CSNG y CDBL.

2.6.1. Función CDBL

Convierte un número de simple precisión (X) a un número de doble precisión. CDBL es la abreviatura de «Convert do Double-precisión» (conversión a doble precisión).

Formato: CDBL(X)

Ejemplos:

```
10 N=1.1
20 PRINT N,CDBL(N)

RUN
 1.1      1.100000023841858
```

```
10 N=345.67
20 PRINT N,CDBL(N)

RUN
 345.67           345.6699829101563
```

Los resultados de CDBL (N) tienen 16 dígitos que es el formato de un número de doble precisión. Obsérvese en el ejemplo N-1.1 que el PC ha añadido una serie de dígitos significativos a la derecha de los ceros y que no tienen relación real con N. En realidad este proceso interno del PC es lógico, si se tiene presente que el número de simple precisión sólo tiene 6 ó 7 dígitos significativos y el redondeo de nuevo a simple precisión producirá de nuevo 1.1.

2.6.2. Función CSNG

Convierte un número de doble precisión (X) a un número de simple precisión. CSNG es la abreviatura de «Convert to SINGLE precisión» (conversión a simple precisión).

Formato: CSNG(X)

Ejemplos:

```
10 N#=1.123456789#
20 PRINT N#;CSNG(N#)

RUN
1.123456789  1.1234567
```

```
10 N#=874.5421332#
20 PRINT N#;CSNG(N#)

RUN
874.3421332  874.5421
```

2.7. Números aleatorios. Función RND/Sentencia RANDOMIZE

Los números aleatorios son números cuyos dígitos no siguen ninguna ley de formación. La aplicación de los números aleatorios son variadas en programación: juegos de azar, estadística, simulación de procesos matemáticos, físicos, industriales, etc.

La función RND genera un número aleatorio comprendido entre 0 y 1. RND es la abreviatura de «RANDOM» (aleatorio).

Formato: | *n* RND(X) |

X se puede omitir en RND

El argumento X es una expresión numérica que afecta al valor devuelto. La salida de RND es un número de siete dígitos que está siempre entre 0 y 1.

Se genera la misma secuencia de números aleatorios cada vez que se ejecuta la función, a menos que el generador de números aleatorios (subrutina interna de la computadora, activada por la sentencia RANDOMIZE, o con el argumento X). En realidad la función RND genera números pseudo-aleatorios que difieren de los números aleatorios en el hecho que el algoritmo que calcula el número pseudo-aleatorio utiliza un valor de partida (semilla) fija. De cualquier forma, denominaremos a todos los números generados por RND, aleatorios.

```
PRINT RND
.6281624
```

Valores del argumento X

X<0	• Se genera una nueva secuencia de números aleatorios.
X>0 X omitido	• Se genera el siguiente número aleatorio de la secuencia.
X=0	• Repite el último número aleatorio generado.

- El siguiente programa genera una lista de cinco números aleatorios:

```
10 FOR I=1 TO 5
20 PRINT RND:
30 NEXT I
99 END

RUN
 .4561316  .9325251  .3517241  .1693156  .2829414
```

- Una modificación utilizando diferente argumento X *(pseudo-aleatorios).*

```
10 '
20 '
30 FOR I=1 TO 5: PRINT RND ; : NEXT I : PRINT
40 FOR I=1 TO 5: PRINT RND ; : NEXT I : PRINT
50 FOR I=1 TO 5: PRINT RND ; : NEXT I : PRINT
60 FOR I=1 TO 5: PRINT RND ; : NEXT I : PRINT
99 END

RUN
 .7151002  .683111  .4821425  .9992938  .6465093
 .1322918  .3692191  .5873315  .1345934  .9348853
 .2194953  .3092079  .2328802  .8951517  .5850341
 .461859  3.686772E-03  .8261403  2.668761E-02  .3744473
Ok
```

```
20 FOR J=1 TO 10
30 A=INT(RND*6)+1
40 PRINT A;
50 NEXT J
99 END

RUN
Random number seed (-32768 to 32767) ?4
 3  4  5  7  3  1  2  2  8  6
```

Número de semilla.

- Obtención de números realmente aleatorios.

`RANDOMIZE TIMER`	*Genera nuevos números de semilla en cada ejecución.*
`RANDOMIZE VAL (RIGHT$(TIME$,2))`	*La semilla de números aleatorios es igual en todo momento al número de segundos indicado por el reloj interno del PC.*

El argumento X suele variar de unas versiones de BASIC a otras, por lo que recogemos las más populares.

MICROSOFT 5	`X>0`	*Número aleatorio entre 0 y 1.*
	`X=0`	*Igual número aleatorio.*
	`X<0`	*Reinicializa la secuencia de números aleatorios.*
TRS-80	`X=0 y X<1`	*Número aleatorio entre 0 y 1.*
	`X>1 y X<32.767`	*Número aleatorio entre 1 y 32.767.*
	`RND (0)`	*Entrega un número real entre 0 y 1.*
	`RND (55)`	*Entrega un número entero mayor que 0 y menor que 56.*
	`RND (55.5)`	*Devuelve un número de igual rango que RND (55), ya que se toma el valor entero del argumento.*
APPLE II (Integer)	`X`	*Proporciona número entero aleatorio entre 0 y valor de X.*
	`RND (1)`	*Proporciona 0.*
	`RND (-2)`	*Proporciona 0 o −1.*
APPLE II (Applesoft)	`X>0`	*Proporciona un número aleatorio diferente entre 0 y 1.*
	`X<0`	*Se realiza la misma secuencia aleatoria siempre.*
	`X=0`	*Proporciona el número aleatorio generado más recientemente.*
COMMODORE	`X=0`	*Entrega un número aleatorio comprendido entre 0 y 1.*
	`X<>0`	*Entrega siempre el mismo número(s) aleatorio(s) y suelen ser imprevisibles.*
DRAGON-32/64	`X=0`	*Entrega un número real entre 0 y 1.*
	`X<>0`	*Entrega un número entero aleatorio entre 1 y x.*
	`RND (121)`	*Entrega un número entero entre 1 y 121.*
IBM/PC	`RND`	*Proporciona número aleatorio entre 0 y 1.*
	`RND (0)`	*Repite el último número aleatorio generado.*
	`RND (1)`	*Genera el siguiente número aleatorio de la secuencia.*

ZX Spectrum	RND RAND	*Proporciona número aleatorio entre 0 y 1. Misma secuencia.*
Amstrad	X > 0 o RND X = 0	*Entrega un número aleatorio entre 0 y 1.* *Proporciona el último número aleatorio generado.*
MSX	X > 0 X < 0	*Entrega un número aleatorio entre 0 y 1.* *Entrega una serie aleatoria que depende de X.* *Proporciona el último número aleatorio generado.*

RESUMEN: Como norma de actuación consideramos, a fin de simplificar, solamente RND(1) cuando se desee obtener números aleatorios y siempre su resultado será un número comprendido entre 0 y 1, remitiendo no obstante al lector al Manual del Usuario de su computadora para su aplicación correcta.

Aplicaciones de RND: *Obtención de números aleatorios.*

`RND (1) *N`	*Entrega un número real comprendido entre 0 y N–1 (inclusive).*
`RND (1) *N+1`	*Entrega un número real comprendido entre 0 y N (inclusive).*
`INT (RND (1)) *N`	*Entrega un número entero comprendido entre 0 y N–1.*
`INT (RND (1) *N)+1`	*Entrega un número entero comprendido entre 0 y N.*
`A+(B-A) *RND (1)`	*Entrega un número comprendido entre A y B (B A).*
`5*RND (1)+10`	*Entrega un número entre 10 y 15.*
`INT (5*RND (1))+10`	*Entrega números enteros 10, 11, 12, 13, 14.*
`INT (2*RND (1))+1`	*Entrega el entero 1 ó el 2.*
`100*RND (1)`	*Entrega un número entre 0 y 100.*

NOTA: *Como aplicación añada una sentencia PRINT a todas las expresiones anteriores y vea que los resultados concuerdan con lo expresado. Dele un valor previamente a N:*

Ejemplo 1:

```
N=10
PRINT RND (1) *N
4.782930
PRINT RND (1) *N+1
9
...
```

Ejemplo: 2

```
100 FOR I=1  TO  5
120 PRINT RND (1)
140 NEXT I
160 END

RUN
 .949653
 .694795
 .298259
 .986284
 .143645
```

2.7.1. Variantes de RND

En diferentes versiones de BASIC se está comenzando a utilizar como variante de RND(X) la función RND y la sentencia RANDOMIZE (o RANDOM) de modo conjunto. De esta forma, la secuencia que genera RND será verdaderamente aleatoria, ya que de no existir RANDOMIZE la secuencia aleatoria sería siempre la misma cada vez que se ejecutase el programa.

Computadoras que trabajan de esta forma son Amstrad, HP-85, HP-87, HP-9845, MICROSOFT, IBM/PC, ZX Spectrum, MSX, etc.

La función RND devuelve un número aleatorio entre 0 y 1. La función RND genera el siguiente número aleatorio basándose en la misma *semilla* (número en el que se basa la secuencia de números aleatorios). Generará la misma secuencia siempre que se ejecute.

Ejemplo:

```
10 FOR I=1 TO 5
20 PRINT INT (RND*100):
30 NEXT I

RUN
 30  40  54  72  16

RUN
 30  40  54  72  16
```

La sentencia RANDOMIZE (o RANDOM) ejecutada antes de RND hace que esta ficción genere números realmente aleatorios.

Formato:

RANDOMIZE (X)

X es opcional según versiones BASIC.

Esta función permite generar una nueva semilla para el generador de números aleatorios. Cada vez que se ejecute RND-RANDOMIZE, evita que se genere en cada ejecución la misma secuencia de números. Si no se pone la serie de números aleatorios, será siempre la misma, como se vio en el ejemplo último:

```
50 RANDOMIZE                 : REM también RANDOMIZE 15
100 FOR I=1 TO 5
120 PRINT INT (RND*100):
140 NEXT I

RUN
 30  40  54  72  36

RUN
 25  17  48  86  94
```

Por la importancia que suponen las computadoras IBM/PC recogemos a continuación el formato de la sentencia RANDOMIZE en la versión BASICA y un ejemplo de aplicación conjunta RANDOMIZE, RND.

Formato:

RANDOMIZE [X]

[ ] símbolo opcional

X expresión numérica entera que se utiliza para generar números aleatorios. X es la semilla del generador que virguará secuencias diferentes de números aleatorios cuando se ejecute RND.

Si se omite X, RANDOMIZE pide al usuario introducir un número de «semilla». Un número de semilla específico entrega siempre el mismo número aleatorio. Un número de semilla diferente producirá un secuencia diferente de números aleatorios. Cambiando el número de semilla X, cambiará el conjunto de números aleatorios generados.

Ejemplos:

```
10 RANDOMIZE
20 FOR I=1 TO 4
30 PRINT RND;
40 NEXT I

RUN
Random number seed (-32768 to 32767) ?2
.7655695  .3558607  .3743453  .1457693

RUN
Random number seed (-32768 to 32767) ?4
.1719568  .5478194  .1387934  .3547851

RUN
Random number seed (-32768 to 32767) ?2
.7655695  .3558607  .3743453  .1457693
```

NOTA 1. *Le recordamos que existen algunas versiones de BASIC, preferentemente las basadas en MICROSOFT, por ejemplo las computadoras TRS-80 y DRAGON-64, en las que la función de números aleatorios es muy especial:*

RND (0) *entrega un número real aleatorio de 0 a 1.*
RND (N) *entrega un número entero aleatorio de 1 a N.*

Ejemplo:

```
PRINT RND (25)          PRINT RND (14)          PRINT RND (5)
 14                      11                      4

PRINT RND (0)
 .14567894
```

NOTA 2: *Como ya se ha mencionado en el párrafo anterior, en el libro consideramos que la generación de números aleatorios entre 0 y 1, y diferentes en cada ejecución, se obtiene con RND (0); pero le recordamos que si su computadora tiene la función RANDOMIZE, para obtener lo mismo deberá escribir la sentencia RANDOMIZE (RANDOM) antes de la ejecución de RND.*

Dado los diferentes matices de la función, le remitimos en su caso particular al manual de su computadora.

2.8. Función SGN

La función SGN entrega el signo de un número. Si el número es negativo, devuelve −1; si es Ø, un 0, y si es positivo, un 1.

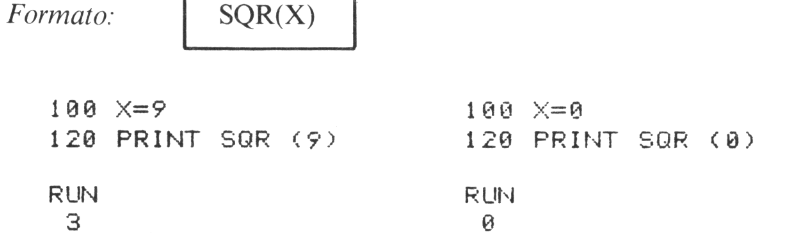
Formato: SGN(X)

Ejemplos:

```
PRINT SGN (-9), SGN (0), SGN (5)
-1         0         1
```

2.9. Función SQR

Devuelve la raíz cuadrada del argumento:



Formato: SQR(X)

```
100 X=9                     100 X=0
120 PRINT SQR (9)           120 PRINT SQR (0)

RUN                         RUN
 3                           0
```

RAIZ CUADRADA DE NUMEROS NEGATIVOS

Como por matemáticas se sabe que no tiene solución real, la computadora imprimirá un mensaje de error:

```
100 X=-4
120 PRINT SQR (X)

RUN
ILLEGAL FUNCTION CALL IN 120
```

Llamada ilegal de función en 120.

3. FUNCIONES LOGARITMICAS Y EXPONENCIALES

Son las representadas por las siglas LOG y EXP, y que permiten cálculos logarítmicos y exponenciales de números o funciones.

3.1. Función LOG

Devuelve el logaritmo natural o neperiano (base *e*) del argumento, con tal que éste sea positivo.

Formato: LOG(X)

Para encontrar el logaritmo de un número en otra base B utilice la fórmula $\log_b (X)=\log_e (X)/\log_e (b)$.

Ejemplo:

Logaritmo decimal de $2=\log_{10} 2=\log_e 2/\log_e 10$ siendo $\log_e 2$ = LOG(2)

LOG(2)/LOG(10) devuelve el logaritmo decimal de 2

LOGARITMO DE NUMEROS NEGATIVOS

Los logaritmos sólo se definen para números positivos.

```
100 PRINT LOG (-4)

RUN
ILLEGAL FUNCTION CALL IN 100
```
(Llamada ilegal de función en 100).

3.2. Función EXP

La función EXP entrega el valor de la potencia de la base *e* (2.718282...) elevada al argumento.



Formato: EXP(X)

Esta función es justo la opuesta de la función LOG.

Por ejemplo, A=EXP(3) es igual que A=2.718282*2.718282*2.718282. El valor del argumento como en las restantes funciones puede ser un número o variable numérica.

ARGUMENTO DEMASIADO GRANDE

Puede ocurrir que el exponente de la función exponencial sea tal, que nos produzca un resultado de desbordamiento (OVERFLOW) al obtener un número mayor que el que soporta el BASIC.

```
PRINT EXP (8)
 2980.96

PRINT EXP (2)
 7.38906

PRINT EXP (128)
OVERFLOW
 1.70141E+38

PRINT EXP (512)
OVERFLOW
 1.70141E+38
```

Se ha superado el valor de 10^{38}, que suele ser el límite superior de trabajo de la mayoría de los BASIC, en los dos últimos ejemplos.

4. FUNCIONES TRIGONOMETRICAS

En este párrafo vamos a tratar un grupo realmente interesante de funciones. Son extremadamente útiles si se han de realizar cálculos matemáticos con figuras geométricas. Los lectores habituados con las funciones trigonométricas no tendrán problema para su comprensión. Para aquellos lectores no familiarizados con la trigonometría, recordaremos sus conceptos fundamentales antes de introducirnos en la programación específica de las funciones trigonométricas.

4.1. Unidades angulares

Los ángulos geométricos normalmente se miden en términos de sexagesimales; en un círculo completo existen 360 grados, 180 grados en un semicírculo y 90 grados en un ángulo recto. Sin embargo la computadora IBM/PC utiliza como unidad angular, los radian en lugar del grado.

¿Qué es un radian? Un *radian* es un ángulo cuya longitud del arco es igual al radio. Un ángulo medio en radianes es igual a la longitud dividida por el radio del círculo. Así por ejemplo al ángulo X, si la longitud es 8 unidades y el radio 5 unidades, tendrá un valor de 8/5 = 1.6 radianes.

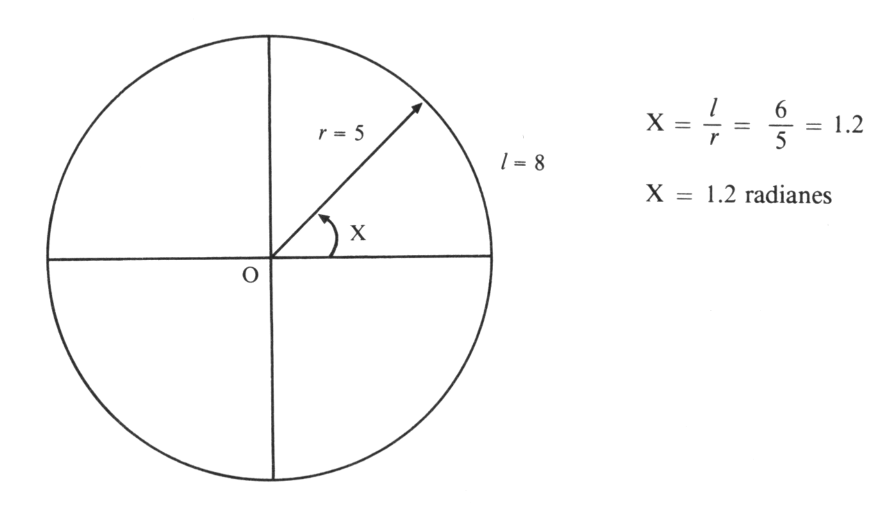


Figura 6.2. Angulos en radianes.

Para poder utilizar las funciones trigonométricas de su computadora necesitará un método que permite convertir grados sexagesimales a radianes y viceversa. Un círculo tiene 2 π (2 Pi grados sexagesimales-360°) (π=pi=3.141592).

360 grados sexagesimales=360° equivalen a 2π radianes
1 grado sexagesimal=1° 1° será 360/2π=180/π radianes=0.01745329
1 radian=180/π grados sexagesimales

La tabla 6.1 representa las fórmulas necesarias para conversión de ángulos sexagesimales a radianes y viceversa.

Tabla 6.1. Conversión grados-radianes/radianes-grados.

Operación	*Fórmula*
Grados sexagesimales a radianes	Radianes=grados*$\frac{\text{Pi}}{180}$ Radianes=grados*0.01745329
Radianes a grados sexagesimales	Grados=radianes*$\frac{180}{\text{Pi}}$ Grados=radianes*57.29578
Regla práctica: *Para convertir grados a radianes, multiplicar los grados por la constante 0.0174532.* *Para convertir radianes a grados multiplicar los radianes por la constante 57.29578.*	

Como se ve, es necesario conocer el valor de π(Pi). Generalmente no se conoce dicho valor por la computadora, con la excepción de algunos modelos, como es el caso de Commodore y Spectrum, que poseen una tecla rotulada con la función PI que al programarla devuelve el valor de Pi. En el resto de los casos será preciso introducir el valor 3.141592..., asignar este valor a una variable.

Las funciones trigonométricas fundamentales basadas en un triángulo rectángulo se representan en la figura 6.3.

Las funciones trigonométricas no son más que relaciones de longitud entre los diferentes lados de un triángulo rectángulo.

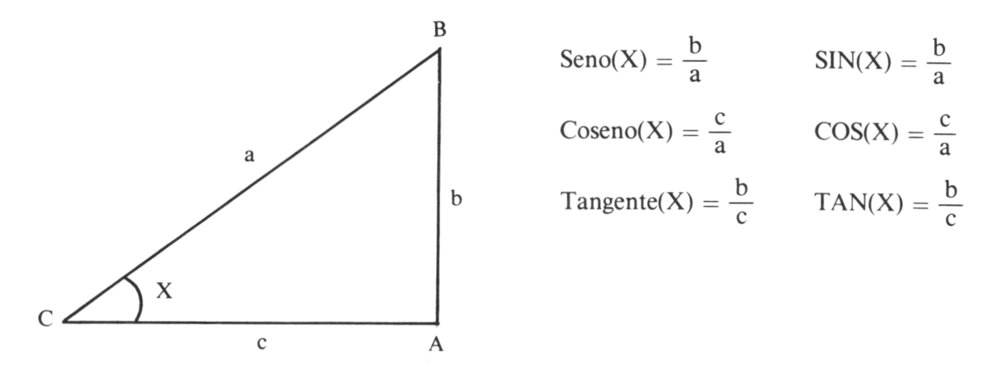


Figura 6.3. Funciones trigonométricas básicas.

Las computadoras actuales proporcionan las funciones trigonométricas más importantes: el ángulo o argumento de la función se expresa en *radianes* normalmente, aunque existen computadoras, como la familia HP-85, HP-87, Amstrad, etc., que permiten seleccionar *radianes, grados sexagesimales o grados centesimales.*

La Tabla 6.2 recoge las funciones trigonométricas más utilizadas.

Las sentencias más usuales son: SIN, COS, TAN y ATN, y suelen estar incorporadas en la mayoría de las computadoras y versiones de BASIC, APPLE II, COMMODORE, TRS-80, «MICROSOFT», «CBASIC», IBM/PC, ZX Spectrum, Amstrad, MSX, etc. Las funciones ASN, ACS, CSC y SEC son típicas de computadoras de tipo científico, como HEWLETT-PACKARD, TEXTRONIX o VAX de Digital.

El argumento X, al igual que en el resto de las funciones, puede ser un variable o expresión numérica.

Tabla 6.2. Tabla de funciones trigonométricas.

Función	*Acción*
SIN (X)	Entrega el seno del argumento X
COS (X)	Entrega el coseno del argumento X
TAN (X)	Entrega la tangente del argumento X
ATN (X)	Entrega el arco tangente del argumento X
ASN (X)	Entrega el arco seno del argumento X
ACS (X)	Entrega el arco coseno del argumento X
CSC (X)	Entrega la cosecante de X
SEC (X)	Entrega la secante de X

4.2. Funciones trigonométricas

Las funciones trigonométricas directas soportadas por la mayoría de las versiones BASIC son:

Formatos:

SIN (X)	*Calcula seno del ángulo X en radianes.*
COS (X)	*Calcula coseno del ángulo X en radianes.*
TAN (X)	*Calcula tangente del ángulo X en radianes.*

Ejemplos:

```
PRINT SIN(5)                    seno de 5 radianes

PRINT SIN(5*0.01745329)         seno de 5 grados
 .871558
```

La función inversa de la tangente es el arco tangente que es el ángulo cuya tangente es el argumento de la función.

Arco tangente (X) = Angulo cuya tangente es igual a X

Formato: ATN(X)

Tabla 6.3. Relación entre las funciones TAN y ATN.

Argumento	*Función*	*Valor de la función*
X en radianes	TAN(X)	$\frac{b}{c} = \frac{\text{lado opuesto}}{\text{lado contiguo}}$
X en radianes	ATN(X)	$\frac{c}{b} = \frac{\text{lado contiguo}}{\text{lado opuesto}}$

Ejemplo: Calcular el seno de 5 grados sexagesimales.

```
5 PI=3.141592
10 X=5*PI/180
20 PRINT SIN (X)

RUN
 .0871558
```

También se podría hacer directamente:

```
5 PRINT SIN (5*3.141592/180)      o bien

5 PRINT SIN (5*0.0174544)

RUN
 .0871558
```

NOTAS:

1. Como ya se ha dicho al principio del párrafo, algunas computadoras pueden trabajar directamente en grados centesimales o sexagesimales. Algunos ejemplos y sus formatos son:

HP-85/HP-87:

Función DTR(X)	Convierte el ángulo X en grados sexagesimales a radianes.
Función RTD(X)	Convierte el ángulo X en radianes a grados sexagesimales.
Función DEG	Pone la computadora en modo *grados sexagesimales* (360 grados sexagesimales en una circunferencia).
Función GRAD	Pone la computadora en modo *grados centesimales* (400 grados centesimales en una circunferencia).
Función RAD	Vuelve la computadora al modo *radianes* (dos π radianes en una circunferencia).

Ejemplos:

```
DEG

PRINT SIN (30)
 .5
PRINT ATN (1)
 45

RAD

PRINT COS (3.141592)
-1
```

Amstrad:

Función DEG	Pone la computadora en el modo grados sexagesimales.
Función RAD	Pone la computadora en el modo radianes.

2. A lo largo del libro se considerará *siempre* que la computadora trabaja *sólo en radianes.*

3. Es preciso tener cuidado con el uso de la tangente, puesto que no está definida cuando el ángulo X vale $\pm\pi/2$, ya que en este caso su valor tendería hacia infinito. Si lo hace, se obtendría en BASIC un mensaje de error tal como éste:

DIVISION BY ZERO

La razón es que la tangente de X se calcula dividiendo SENO (X) por COSENO (X) y COSENO de $\pm\pi/2$ es 0.

Evite obtener la tangente de $\pm\pi/2$ radianes o $\pm 90°$, o añada un pequeño error al valor del argumento que impida el valor exacto.

4.2.1. Programas ejemplo

1.
```
10 PRINT "INTRODUCIR UN ANGULO (RADIANES)"
20 INPUT R
30 A=SIN (R)
40 PRINT "SENO DE": R: "RADIANES", A
50 GOTO 20
```

```
RUN
INTRODUCIR UN ANGULO (RADIANES)
? 5
SENO DE 5 RADIANES                -0.9589243
INTRODUCIR UN ANGULO (RADIANES)
? 1
SENO DE 1 RADIANES                -0.8414710
```

2.
```
100 PRINT "INTRODUCIR UN ANGULO (RADIANES)"
120 INPUT R
140 A=COS (R)
160 PRINT "COSENO DE": R: "RADIANES": A
180 GOTO 100

RUN
INTRODUCIR UN ANGULO (RADIANES)
? 1
COSENO DE 1 RADIANES 0.540302
INTRODUCIR UN ANGULO (RADIANES)
? 6
COSENO DE 6 RADIANES 0.96017
```

3.
```
10 INPUT "Introducir angulo en grados:",GRADOS
20 PI=3.141592
30 RADIANES=GRADOS*PI/180
40 PRINT SIN(RADIANES),COS(RADIANES),TAN(RADIANES)
99 END

RUN
Introducir angulo en grados:60
 .8660254       .5000001        1.732051
```

4.
```
10 INPUT "Introducir el valor de la tangente:",TANGENTE
20 RADIAN=ATN(TANGENTE)
30 GRADOS=57.29578*RADIAN
40 PRINT GRADOS,RADIAN
99 END

RUN
Introducir el valor de la tangente:1
 8.999426       1.570696
```

4.2.2. Notas sobre la precisión de las funciones trigonométricas

Los cálculos se realizan normalmente en simple precisión, y según los valores del ángulo se pueden obtener algunos errores en los cálculos, que vamos a tratar de analizar.

```
PRINT COS(60*0.01745329)        Cálculo coseno de 60°
 .5000001
```

El coseno de 60 grados es un valor exacto 0.5 y, sin embargo, se obtiene el valor .5000001, la diferencia corresponde a la precisión de los cálculos en simple precisión.

```
PRINT TAN(90*0.01745329)        Cálculo tangente de 90°
 -2670177
```

La tangente de 90° es infinito y se obtiene un número negativo muy grande 2670177, pero no igual a infinito que por supuesto es mucho mayor.

```
PRINT SIN(90*0.01745329)      Cálculo coseno de 90°
 -3.74507E-7
```

se obtiene un número muy pequeño que no es cero.

Si necesita una gran precisión deberá trabajar con el sistema operativo DOS 2.0 que proporciona una doble precisión trigonométrica de 16 dígitos.

5. FUNCIONES DEFINIDAS POR EL USUARIO

Hasta ahora se han definido un cierto número de funciones numéricas tales como SIN, COS, EXP, RND, etc. Sin embargo, puede que no sean suficientes para nuestros cálculos y se desee crear otras.

Así, por ejemplo, en matemáticas es corriente utilizar funciones parecidas a

$$f(x)=x^3-4x^2+5$$

En ellas es preciso calcular el valor de la función f(x) para diferentes valores de la variable x. Mediante la función DEF FN, nos permite programar la función f(x) de igual modo que las funciones incorporadas al BASIC ya mencionadas.

5.1. Función DEF FN

La funció¡n DEF FN permite que sean definidas y utilizadas funciones de aplicaciones especiales.

Para conseguir lo anterior, distinguiremos entre:

— La definición de la función.
— Su utilización.

Formato: | *n* DEF FN *variable (arg)=expr* |

n	Número de línea.
variable	Nombre de una variable numérica válida.
arg	Nombre de la variable o argumento de la función.
expr	Expresión que representa una función matemática.

Ejemplo 1: Representar la función X^2+1.

```
DEF FNA (X)=X^2+1
```

La variable o argumento encerrado entre paréntesis debe concordar con la variable usada en la expresión a la derecha del signo igual. Estas variables se les conoce como *variables «falsas».*

En nuestro ejemplo la expresión X2+1 está definida como FNA. FN (una abreviatura de la palabra FuNction) se utiliza en las sentencias o funciones DEF por cualquier variable numérica legal. Así, A en este ejemplo es la función X^2-1, pero se podría haber utilizado cualquier variable.

Ejemplo 2: Representar la función $f(x)=X^2 \cdot \cos(x)+5$.

```
DEF FNH (X)=X^2*COS (X)+5
```

UTILIZACION DE UNA FUNCION

La operación almacenada en la función FN por la sentencia DEF se puede utilizar para manipular cualquier número o variable numérica.

Ejemplo 1:

```
10 DEF FNA (X)=X^2+1
20 R=2
30 PRINT FNA (R)

RUN
 5
```

La función en este ejemplo se llama «A» (FNA) y se le asigna la ecuación X2+1 en la línea 10. Cada vez que FNA se ejecute, es como si se sustituyera la variable falsa X por R o, lo que es igual, FNA(R)=FNA(2)=22+1=5.

Ejemplo 2:

```
100 B=6: C=5
120 DEF FNA (V)=V/B+C
140 Z=FNA (3)
160 PRINT Z

RUN
 5.5
```

5.1.1. Ejemplos de funciones FN

a) Redondeo de un número con dos decimales:

```
100 DEF FNA (X)=INT (X*100+0.5)/100
120 PRINT FNA (4.57264): FNA (31.413)

RUN
 4.57  31.41
```

b) Obtención de diferentes valores de la función $f(x)=x^2-4$.

```
100 DEF FNA (X)=X^2-4
120 X=2
140 B=FNA (0)
160 C=FNA (X)
180 PRINT B; C

RUN
-4  0
```

c) Empleo de funciones con variables normales y «falsas».

```
100 A=5: B=2
200 DEF FNH (X)=A*X+5
220 PRINT FNH (B)
240 A=1
260 PRINT FNH (B)

RUN
 15
 7
```

d) Redondeo de números con N cifras decimales.

```
100 DEF FNA (X)=INT (X*10^N+0.5)/10^N
120 X=134.1754
140 FOR  N=1  TO  3
160 PRINT FNA (X)
180 NEXT N

RUN
 134.2
 134.18
 134.175
```

POSIBLES ERRORES

La función DEF debe estar definida antes de su utilización ya que de lo contrario se produciría un mensaje de error:

```
UNDEFINED USER FUNCTION IN N
```
Función de usuario no definida en N

```
100 A=5
120 PRINT FNA (X)
140 DEF FNA (Z)=Z*Z-1

RUN
UNDEFINED USER FUNCTION IN 120
```

5.1.2. *Variante de la función* DEF FN

Algunas computadoras permiten más de una variable en la expresión definida. Cada una de estas variables se debe listar después de la variable:

Formato:

DEF FN *variable* $(arg_1, arg_2, ...)$

Ejemplo 1:

```
100 REM DEF MULTIPLE
120 DEF FNA (X.Y)=(X+Y)/2
140 INPUT "INTRODUCIR 2 NUMEROS"; X, Y
160 B=FNA (X, Y)
180 PRINT "LA MEDIA ES"; B
200 END

RUN
INTRODUCIR 2 NUMEROS? 30, 50
LA MEDIA ES 40
```

Ejemplo 2:

```
100 DEF FN H (X, Y, Z)=(X+Y+Z)/3
120 A=15: B=30: C=45
140 PRINT FN H (A, B, C)

RUN
 30
```

PROGRAMAS RESUELTOS

1. *Aplicación de* SQR, LOG, EXP.

```
100 REM MAQUINA ALGEBRAICA
120 PRINT "DEME UN NUMERO"
140 PRINT "Y CALCULARE SU RAIZ CUADRADA"
150 PRINT "LOGARITMO Y EXPONENCIAL"
160 PRINT
200 INPUT "NUMERO": N
210 PRINT
220 PRINT "RAIZ CUADRADA", "LOGARITMO", "EXPONENCIAL"
240 PRINT SQR (N), LOG (N), EXP (N)
250 GOTO 120
300 END

RUN
DEME UN NUMERO
Y CALCULARE SU RAIZ CUADRADA
LOGARITMO Y EXPONENCIAL

NUMERO? 25

RAIZ CUADRADA   LOGARITMO        EXPONENCIAL
 5               3.21888          7.20049E+10
```

2. *Aplicación de* INT *y* RND.

```
100 REM NUMEROS ALEATORIOS CON UNA O DOS CIFRAS ENTERAS
120 REM Y TRES DECIMALES REDONDEADAS
140 FOR K=1  TO  10
160 A=RND (1) *100
180 B=INT (A*1000+0.5)/1000
200 PRINT B.
220 NEXT K
240 END

RUN
 90.689          24.934          47.644          56.33
 19.791          3.707           50.411          82.771
 55.553          40.814
```

3. *Aplicación de* RND.

```
100 INPUT "CUANTOS NUMEROS ALEATORIOS DE 2 CIFRAS": N
120 PRINT
140 FOR I=1  TO  N
160 PRINT INT (100*RND (1));
180 NEXT I
200 PRINT
```

```
220 GOTO 100
999 END

RUN
?CUANTOS NUMEROS ALEATORIOS DE 2 CIFRAS? 10

 76  74  73  70  46  73  76  35  90  47
```

4. *Aplicación de* DEF FN.

```
100 DEF FNH (X)=INT (X*1000+0.5)/1000
120 PRINT "REDONDEO DE NUMEROS CON TRES DECIMALES"
140 INPUT N
160 PRINT N; "REDONDEADO ES"; FNH (N)
180 IF N=0 THEN 400
200 GOTO 120
400 END

RUN
REDONDEO DE NUMEROS CON TRES DECIMALES
? 31.47645
 31.4765 REDONDEADO ES 31.476
```

5. *Aplicación de funciones trigonométricas y logarítmicas.*

```
100 A=2
120 B=5
140 C=10
160 PRINT SIN (A); COS (B); LOG (C)

RUN
 .909298  .283662  2.30259
```

6. *Aplicación de la función* INT.

```
100 A=34.5674
120 B=45.3221
140 A1=INT (A)
160 B1=INT (B)
180 PRINT A; B; A1; B1

RUN
 34.5674  45.3221  34  45
```

7. *Aplicación de la función exponencial.*

```
100 A=4.5
120 B=EXP (A)
140 C=2.718281^A
160 PRINT A; B; C

RUN
 4.5  90.0171  90.017
```

8. *Aplicación de la función* RND.

```
100 FOR I=1  TO  6
110 PRINT RND (1)
130 NEXT I
140 END

RUN
 .234235
 .310937
 .677161
 .387965
 .526114
 .645637
```

9. *Aplicación de la función* RND.

```
10 FOR I=1 TO 8
20 PRINT INT (RND (1) * 100+1);
30 NEXT I

RUN
 72  4  8  15  99  14  100  47
```

10. *Aplicación de la función* RND.

Obtención de números aleatorios hasta encontrar el número 15.

```
100 PRINT "Genero numeros aleatorios entre 1 y 50"
120 PRINT "Me detengo si obtengo el 15"
140 PRINT
160 R=INT (RND (1) *50 + 1)
180 PRINT R
200 IF R<>15 THEN 160
220 PRINT "Caspitas! Acerte"
240 PRINT "el numero 15"
260 END


RUN
Genero numeros aleatorios entre 1 y 50
Me detengo si obtengo el 15

 20
 33
 8
 28
 15
Caspitas! Acerte
el numero 15
```

11. *Aplicación de la función* RND.

El siguiente programa adivina el número entero que se introdujo mediante el teclado.

```
10 REM Adivinanza de numeros
20 PRINT "Puedo adivinar el numero"
```

```
30 PRINT "que introduzca por teclado entre 1 y 100"
40 INPUT "Numero"; N
50 IF N>100 OR N<1 THEN PRINT "No adivino
   numeros mayores que 100 ni menores que 1":GOTO 40
60 R=INT (RND (1) * 100 + 1)
70 PRINT R
80 IF R<>N THEN 60
100 PRINT "Albricias el numero es"; R


RUN

Puedo adivinar el numero
que introduzca por teclado entre 1 y 100
Numero? 54
 59
 7
 55
 14
 54
Albricias el numero es 54
```

12. *Aplicación de* SQR.

Resolución de una ecuación de segundo grado.

Sea: $Ax^2+Bx+C=0$.

Recordemos que si A=0 es una ecuación de primer grado; si C es 0 en este caso es indeterminada la solución; si el descriminante B^2-4AC es <0, las soluciones son imaginarias.

Las raíces reales son:

$$x_1,x_2=(-B\pm\sqrt{B^2-4AC})/2A$$

Por tanto, el programa quedaría así:

```
10 REM Resolucion ecuacion segundo grado
20 INPUT "Coeficientes A, B, C"; A, B, C
30 IF A=0 THEN 200
40 IF C=0 THEN 300
50 D=B^2-4*A*C
60 IF D<0 THEN 100
70 X1=(-B+SQR(D))/(2*A)
80 X2=(-B-SQR(D))/(2*A)
90 PRINT "X1=";X1, "X2=";X2
95 GOTO 400
100 REM D menor que cero
110 PRINT "Raices imaginarias"
120 GOTO 400
200 REM A=0
210 PRINT "Ecuacion de primer grado"
220 PRINT "X="; -C/B
230 GOTO 400
300 REM C=0
310 PRINT "X1= 0", "X2="; -B/A
400 END
```

```
RUN
Coeficientes A, B, C? 2, 3, 4
Raices imaginarias

RUN
Coeficientes A, B, C? 1, 4, 3
X1=-1          X2=-3
```

13. *Aplicación de las funciones trigonométricas.*

Obtención de la Tabla de funciones trigonométricas, seno, coseno y tangente para los ángulos de 2 a 30° sexagesimales de 4 en 4 grados.

```
100 FOR I=2 TO 30 STEP 4
120  A=I*PI/180
140 PRINT SIN(A), COS(A), TAN(A)
160 NEXT I
```

```
RUN
 .0348995        .999391         .0349208
 .104528         .994522         .105104
 .173648         .984808         .176327
 .241922         .970296         .249328
 .309017         .951057         .32492
 .374607         .927184         .404026
 .438371         .898794         .487732
 .5              .866025         .57735
```

Si su computadora no tiene la función PI o π deberá introducir una línea

```
10 PI=3.141592
```

14. *Aplicación de las funciones exponencial y logarítmica.*

Obtener una tabla que obtenga el logaritmo y los exponenciales de los números 10 a 20.

```
100 FOR I=10 TO 20
120 PRINT LOG (I), EXP (I)
140 NEXT I
```

```
RUN
 2.30259         22026.5
 2.3979          59874.1
 2.48491         162755
 2.56495         442413
 2.63906         1.2026E+06
 2.70805         3.26902E+06
 2.77259         8.88611E+06
 2.83321         2.41549E+07
 2.89037         6.56599E+07
 2.94444         1.78482E+08
 2.99573         4.85165E+08
```

15. *Aplicación de la función* INT.

Deducir si un número N introducido por teclado es primo.

Una regla aritmética para su deducción es comprobar que todos los enteros a partir de 2 hasta el valor de la raíz cuadrada de ese número, N, no son divisores de ese número. Como regla práctica basta comprobar si la parte entera de la división de N por todos los números comprendidos entre 2 y N es igual a dicha división.

```
100 REM Numeros primos
120 INPUT "Numero"; N
140 FOR I=2 TO SQR (N)
160 IF N/I=INT (N/I) THEN 240
180 NEXT I
200 PRINT N;"es primo"
220 GOTO 120
240 PRINT N;"es divisible por";I
300 GOTO 120
400 END


RUN
Numero? 7
 7 es primo
Numero? 14
 14 es divisible por 2
```

16. *Aplicación de las funciones* FIX, CINT *y* ABS.

Este programa convierte la hora del día en formato decimal al formato clásico de horas y minutos.

```
10 INPUT "Introducir la hora en forma decimal:",TIEMPO
20 HORAS=FIX(TIEMPO)
30 MINUTOS=CINT(ABS(TIEMPO-HORAS)*60)
40 PRINT "La hora actual es "HORAS "horas y "MINUTOS "minutos."
99 END

RUN
Introducir la hora en forma decimal: 4.362
La hora actual es 4 horas y 22 minutos.
OK
```

Las líneas 20 y 30 realizan los cálculos necesarios mediante las funciones FIX y CINT. Las funciones ABS aseguran que los minutos son siempre positivos y CINT redondea los minutos al entero más próximo. En la línea 20 se puede utilizar INT en lugar de FIX, pero en el caso de introducir un número negativo se obtendría un error y por ello es preferible la función FIX.

PREGUNTAS Y PROGRAMAS DE REPASO

1. ¿En qué partes se puede utilizar una función BASIC?

2. Escribir un programa que permita calcular la longitud de la hipotenusa de un triángulo a partir de las longitudes de los catetos.

   NOTA: $a^2=b^2+c^2$ a, hipotenusa, b y c, catetos.

3. Diga qué sucede al ejecutar este programa:

```
10 INPUT N
20 IF N/2 = INT (N/2) THEN 50
30 PRINT N; "No es entero"
40 GOTO 10
50 PRINT N; "Es entero"
60 GOTO 10
100 END
```

4. Escribir un programa que permita introducir por teclado dos ángulos A, B, y se calcule una tabla cuyas columnas sean SENO A, COS B, SEN (A+B), COS (A+B).

   NOTA: sen (A+B)=sen A. cos B+cos A. sen B
   cos (A+B)=cos A. cos B+sen A. sen B

5. ¿Qué sucede al ejecutar el programa?:

```
10 X=16
20 FOR I=1 TO 5
30 PRINT SQR (X)
40 X=-X
50 NEXT I
100 END
```

6. ¿Qué sucede al ejecutar el programa?:

```
10 C=3.1415926/180
20 FOR X=0 TO 90
30 X1=X*C
40 PRINT X; " "; SIN (X1); " "; COS (X1); " "; TAN (X1)
50 NEXT X
100 END
```

7. Ejecute los programas siguientes y vea sus resultados:

```
5 REM Programa A
10 INPUT A, N
20 W=SQR(A)
```

```
30 W=INT (W*10^N+0.5)/10^N
40 PRINT W
............................
5 REM Programa B
10 INPUT A, N
20 PRINT INT (SQR (A) * 10^N+0.5)/10^N
```

8. Diga qué se obtendría tras la ejecución de cada una de las siguientes sentencias:

```
IF ABS (X-Y)<0.001 THEN PRINT "LA APROXIMACION
ES MENOR DE 0.001"
INT (1.742051*10^4+0.5)/10^4
INT (1.742051)/1000
PRINT INT (SQR (3) * 10^4)/10^4
```

9. Realice los siguientes programas:

```
10 INPUT C
20 IF C/2 = INT (C/2) THEN PRINT C, "PAR":GOTO 40
30 PRINT C, "IMPAR"
40 END
```

10. Explique detenidamente el concepto de número «aleatorio» y exprese todas las posibles variaciones que la programación lleva consigo.

11. Estudie el programa siguiente y analice su resultado:

```
100 A=RND (1)
120 H=INT (A*5) + 3
140 PRINT H
160 END
```

PROGRAMAS PROPUESTOS

1. Realizar un programa que permita simular una partida de dados de modo que cada jugador lance un dado diez veces, imprimiendo el resultado de la tirada de cada jugador. El número de jugadores es 5.

2. Escribir un programa que permita efectuar tiradas de monedas al aire y que realice los puntos siguientes:

   a) Introducción por teclado del número deseado de lanzamientos.
   b) Impresión del resultado de los lanzamientos.
   c) Impresión del número total de «caras» y de «cruces» que se han obtenido.

3. Escribir un programa que imprima una tabla de las funciones trigonométricas de 0 a 90 de 1.º en 1.º (grados sexagesimales, y con el siguiente formato):

Angulo	*Seno*	*Coseno*	*Tangente*
0	0	1	0
—	—	—	—
...	...	...	...

4. Se desea obtener los diferentes valores del polinomio Ax^3+Bx^2+Cx+D, de modo que para cada cuarterna de valores A, B, C, D, se ha de calcular el valor correspondiente del polinomio considernado que x toma los valores 1 a 25.

   NOTA: El problema se ha de resolver aplicando DEF FN.

5. Trate de realizar un programa para que un niño de siete años practique la multiplicación. El niño podrá elegir los factores de un solo dígito por teclado y luego deberá introducir la solución. Si es correcta, la computadora le contestará ¡ENHORABUENA!, y en caso contrario VUELVE A EMPEZAR, TE HAS EQUIVOCADO.

6. Obtener un programa que obtenga la lista de los números primos comprendidos entre 1 y 500.

7. Realice un programa que permita obtener diez números al azar de una lotería que contiene papeletas numeradas del 1 al 5000.

8. Realice un programa que obtenga 100 números aleatorios y calcule su media aritmética (suma de los 100 números dividida por 100).

9. Realice un programa que imprima una tabla de logaritmos decimales (base 10) comprendidos entre 1 y 100.

CAPITULO 7

Entrada-salida de datos

0. INTRODUCCION

A lo largo de esta lección se tratarán las sentencias de entrada-salida que permiten la introducción de datos que son incorporados al programa mediante las sentencias READ-DATA-RESTORE, dado que la sentencia INPUT que permite la entrada de datos por teclado ya se trató en el Capítulo 3.

A la terminación del tema, el lector podrá trabajar adecuadamente con las sentencias siguientes:

— READ
— DATA
— RESTORE
— TAB
— POS
— SPC
— PRINT USING
— IMAGE

1. READ-DATA

La introducción de datos en la memoria de la computadora se realiza mediante sentencias INPUT y READ/DATA que permiten el tratamiento de datos numéricos y alfanuméricos, como ya se ha visto, mediante la sentencia INPUT la introducción de datos tiene un inconveniente: el programa se detiene y espera hasta que se introducen los datos solicitados.

Mediante las sentencias READ/DATA se pueden introducir datos de modo permanente en un programa sin necesidad de tener que hacerlo por teclado durante la ejecución.

La sentencia READ permite la asignación de valores a variables numéricas o de cadena.

Formato: | *n* READ *variable*$_1$, *variable*$_2$,... |

n	Número de línea.
variable$_1$	Variable numérica o de cadena legal.
variable$_2$	Variable numérica o de cadena legal.

Cada sentencia READ puede definir una o varias variables de cualquier tipo separadas por una coma.

Ejemplos:

```
10 READ A$
20 READ A
30 READ A$, B$, C, D, E
```

La sentencia DATA contiene los valores que asignaron a las sentencias READ.

Formato: *n* DATA *constante numérica o de cadena,...*

Ejemplos:

```
100 DATA 12, 45, 324
120 DATA "MORTIMER", "MCKOY", "FLANAGAN"
140 DATA "JOE", 45, 7.34
```

La palabra DATA es seguida por las respectivas constantes numéricas o alfanuméricas que se asignarán a las variables contenidas en las sentencias READ.

Las sentencias READ y DATA van ligadas de modo que a cada variable de una sentencia READ se corresponde un dato de la sentencia DATA.

Así, una sentencia READ-DATA equivale a una asignación.

```
10 DATA 6, 13
20 READ A, B
```

Estas líneas equivalen a:

```
10 A=6
20 B=13
```

```
10 DATA "AZUL", 12, "SI", 3.5
20 READ A$, B, C$, D
```

Estas líneas equivalen a:

```
10 A$="AZUL"
20 B=12
30 C$="SI"
40 D=3.5
```

MODO DE FUNCIONAMIENTO READ/DATA

Cuando un programa llega a una sentencia READ, se leen todas las variables que vienen en ella y a continuación se dirige hacia la sentencia DATA más inmediata, de modo que se van asignando a cada variable de READ un dato recogido en la sentencia DATA. Un ejemplo práctico es el siguiente:

```
10 READ A, B, C
20 PRINT
30 PRINT A, B, C
100 DATA 4.25, 7, 8.24
```

```
RUN
 4.25            7              8.24
```

Al ejecutarse este programa se asignan los siguientes valores:

```
100 DATA 45, 61, 23, 40, 300
120 READ A
140 PRINT 2*A
160 GOTO 120
200 END
```

La primera sentencia ejecutada asigna a la variable A el valor 45. La línea 140 imprime 2*A=90. A continuación vuelve a la línea 120, en donde se vuelve a leer de nuevo la variable A, en cuyo caso se toma el valor 61 que sustituye al primer valor 45. Al ejecutarse el programa se obtendrá:

```
 90
 122
 46
 80
 600

OUT OF DATA IN 120
```

Este mensaje significa que se han terminado los datos de la línea DATA y en consecuencia no se pueden seguir leyendo. La sentencia GOTO envía el programa a la línea 120 de modo que cuando se hayan leído todos los datos de la línea 10, se presentará el mensaje OUT OF DATA que significa «faltan datos» o bien «datos agotados».

REGLAS DE APLICACION DE READ/DATA

1. Los datos indicados por medio de sentencias READ/DATA forman parte del programa, en contraste con los introducidos por medio de la sentencia INPUT. Por consiguiente los datos se almacenarán en el programa y se asignarán a la variable correspondiente cuando se ejecute el mismo, no pudiendo modificarse más que cuando se modifiquen las(s) sentencia(s) DATA.
2. Las sentencias DATA pueden aparecer en cualquier parte del programa. Pueden venir asociadas después de la sentencia READ asociada, o intercaladas a lo largo del programa. Sin embargo, se recomienda que se disponga inmediatamente antes o después de su sentencia READ o bien agrupadas todas al final del programa (a veces al principio) con lo que la localización y lectura de las mismas será más fácil.
3. Los datos en una sentencia DATA pueden ser cadenas, números o combinación de ambas.
4. Los datos de una sentencia DATA deben estar separados por comas, excepto el último dato.
5. Las cadenas que contengan comas o que comiencen con espacios en blanco exige que se representen entre comillas. (Esta regla no la cumplen todos los BASIC.)
6. El tipo de variable de una línea READ debe concordar con los datos que se leen en la sentencia DATA (en otras palabras, no se puede leer una cadena en una variable numérica).
7. No hay restricciones en el número de datos que una línea DATA pueda contener.
8. Los números en las sentencias DATAS no exigen nunca comillas.
9. La sentencia DATA es no ejecutable.

Ejemplos:

1) *Orden de la sentencia DATA.*

```
100 FOR I=1 TO 5
110 READ X
120 DATA 5, -10, 20
130 PRINT X
140 DATA 45, 32
150 NEXT I

RUN
 5
-10
 20
 45
 32
```

En la realización del bucle I a la variable X se le han asignado los valores 5, – 10, 20, 45, 32.

En este programa se podrían haber sustituido las líneas 120 y 140 por una única línea 120 DATA 5, –10, 20, 45, 32.

Un nuevo programa que muestra otra posición diferente de la sentencia DATA es:

```
10 DATA 30, 100, 225
20 FOR I=1 TO 5
30 READ A
40 PRINT A
50 NEXT I
60 END

RUN
 30
 100
 225
OUT OF DATA IN 30
```

En la primera iteración del bucle a la variable A se le asigna el valor 30; en la segunda iteración el valor 100, en la tercera 225, y en la cuarta iteración nos encontramos con que no existen más valores en la línea 10 que se puedan leer, o lo que es igual se han leído todos los valores; ello representa una anomalía que se refleja en el programa.

`OUT OF DATA IN 30` *(Faltan datos en la línea 30).*

2) *Concordancia de datos en READ y DATA.*

```
100 READ A$, B, C

200 DATA FLANAGAN, 45, 37
```

Sin embargo no sería correcto:

```
100 READ A$, B, C
200 DATA 45, 37, A$
```

3) *Escritura de cadenas:*

— sin comillas,

```
100 DATA DUPONT, MARTINEZ, GARCIA
200 READ A$, B$, C$
```

— con comillas (algunas computadoras exigen las cadenas con comillas,

```
100 DATA "DUPONT", "MARTINEZ", "GARCIA"
```

Las varibles A$, B$ y C$ toman los valores DUPONT, MARTINEZ y GARCIA.

4) *Cantidad de datos por línea opcional.*

Se puede escribir:

```
100 DATA 1005, 2034, 432, 5, 72
```

o

```
100 DATA 1005, 2034
200 DATA 432, 5, 72
```

5) *Mezcla de datos numéricos y alfanuméricos en una línea.*

```
100 DATA "AZUL", 123, "VERDE", 1972, 45
120 READ A$, B, C$, D, E
```

6) *Igual número de variables en la sentencia READ que constantes en la sentencia DATA.*

En caso de no cumplirse esta exigencia aparecerá un mensaje de error OUT OF DATA IN *n* (FALTAN DATOS EN *n*, siendo *n* número de línea de la sentencia READ correspondiente) que significa que se intentó leer con una sentencia READ más datos de los que hay en sentencias DATA.

```
10 DATA 40, 50, 60, 30
20 READ A, B, C, D, E

RUN
OUT OF DATA IN 20
```

se leen cinco variables y sólo existen cuatro constantes.

7) *La posición de la sentencia DATA es arbitraria.*

```
10 DATA 40, 50
20 READ A, B, C
30 DATA 100, 200, 300
40 READ E, F, G, H, I
50 DATA 400, 600, 800
```

2. EJERCICIOS PRACTICOS CON SENTENCIAS READ/DATA

Al objeto de afianzar la comprensión de las sentencias READ/DATA vamos a ver una serie de diferentes ejemplos que nos permita su fácil manejo:

a)

```
10 DATA 20, 45, 32, 90, 50
20 READ N
30 PRINT N
40 GOTO 20
50 END

RUN
 20
 45
 32
 90
 50
OUT OF DATA IN 20
```

El programa, al ejecutar RUN, en la línea 20 lee N y le asigna el primer valor de DATA, *20*; a continuación imprime dicho valor y bifurca a la línea 20, por lo que se vuelve a leer un nuevo dato de la línea 10, *45* en este caso; en la quinta iteración se lee *50* y se imprimen, retornando el programa de nuevo a la línea 20 que vuelve a leer N, pero en este caso no le puede asignar ningún valor, dado que ya se han leído y asignado las cinco constantes de la línea DATA. Por consiguiente, aparece un mensaje de error OUT OF DATA IN 10.

b)

```
10 READ A, B
20 PRINT A, B, A*B
30 GOTO 10
40 DATA 42.5, 22, 43, 21
50 DATA 4, 2, 3, 7
60 END

RUN
 42.5           22              935
 43             21              903
 4              2               8
 3              7               21
OUT OF DATA IN 10
```

Faltan datos en las líneas DATA cuando se han leído los 10 datos iniciales.

c) Programa de aplicación típico.

Localizar e imprimir los nombres de los menores de dieciocho años de una lista de datos específica.

```
100 PRINT "Nombre", "Edad"
120 READ N$
140 IF N$="FIN" THEN PRINT "Fin de la lista":GOTO 460
160 READ E
180 IF E<18 THEN PRINT N$, E
200 GOTO 120
300 REM Lista de datos
320 DATA "Luis Garcia", 30
340 DATA "Jose Gil", 25
```

```
360 DATA "McKoy", 15
380 DATA "Mortimer", 16
400 DATA "Flanagan", 17
420 DATA "Jesus Gismera", 16
440 DATA "FIN"
460 END

RUN
Nombre              Edad
McKoy                15
Mortimer             16
Flanagan             17
Jesus Gismera        16
Fin de la lista
```

Una lista similar a la anterior se llama *«fichero»*. En la línea 440 se introduce un dato *fin de fichero* «FIN» que permitirá al ser leída por la sentencia de la línea 160 imprimir «FIN DE LA LISTA» y terminar.

Este procedimiento es el más utilizado para dar por finalizada una lectura de datos y evitar la aparición del incómodo mensaje OUT OF DATA IN LINE *n.*

d) Tratamiento de cadenas

```
100 READ A$
120 READ B$
140 PRINT A$+B$
160 DATA "QUE GRANDE ERES "
180 DATA "ENANO"

RUN
QUE GRANDES ERES ENANO
```

3. RECOMENDACIONES PRACTICAS PARA USO DE READ/DATA

Revisemos de modo práctico las aplicaciones más corrientes de las sentencias READ/DATA, introduciendo el concepto elemental de *fichero, marca de fin de fichero,* etc.

3.1. Datos permanentes

Supongamos que desea realizar un programa que calcule las calificaciones medias de los alumnos de una clase, pero de forma que el número y los nombres de los alumnos figuren de modo permanente en programa; por ejemplo, doce alumnos cuyos apellidos son: MORTIMER, MACKOY, FLANAGAN, LOPEZ, PEREZ RUIZ, GARCIA, MARTINEZ, GIL, GOMEZ, GILA, MARIN. Tanto 12 como los 12 apellidos figurarán en sentencias DATA. Las notas o calificaciones parciales se introducirán mediante teclado. (El conjunto de los 12 nombres forman un *fichero* elemental).

```
100 REM Notas medias
120 READ N
140 FOR J=1 TO N
160 READ A$
180 PRINT "Cuantas notas del alumno "; A$
200 INPUT C
210 FOR K=1 TO C
```

```
220 PRINT "Nota numero";K
240 INPUT A
260 S=S+A
280 NEXT K
300 M=S/C
320 PRINT "Media alumno "; A$; " ="; M
340 NEXT J
400 DATA 12
420 DATA "Mortimer", "McKoy", "Flanagan", "Lopez"
440 DATA "Perez", "Garcia", "Ruiz", "Martinez",
"Gil", "Gomez", "Gila"
460 DATA "Marin"

RUN
Cuantas notas del alumno Mortimer
? 3
Nota numero 1
? 5
Nota numero 2
? 7
Nota numero 3
? 9.30
Media alumno Mortimer = 7.1
```

NOTA: *Si desea cambiar los nombres de los alumnos bastará hacerlo en las sentencias DATA correspondientes.*

3.2. Colocación de constantes en sentencias DATA

Las sentencias DATA se pueden colocar en cualquier lugar.

```
100 FOR I=1 TO 5
120 READ N
140 DATA 4, -10, 15
160 PRINT N
180 DATA 20, 25
200 NEXT I

RUN
 4
-10
 15
 20
 25
```

Se pueden sustituir las líneas 140 y 140 por una sola 160 tal como:

```
140 DATA 4, -10, 15, 20, 25
```

3.3. Evitar confundir cadenas y constantes numéricas

La lectura de datos debe corresponderse en número y clase en las sentencias READ/DATA.

```
100 DATA "MORTIMER", "FLANAGAN"
120 READ N ,B$
```

Al acceder la línea READ al primer valor en la línea DATA comprueba que es una cadena de caracteres en vez de una constante como se previó, lo que producirá una señal de error similar a:

```
SYNTAX ERROR IN 100
```

3.4. ¿Es necesario poner la cadenas entre comillas?

Según los modelos de computadoras, puede ser o no necesario.

Caso a:

```
100 DATA 25
110 READ N$
```

a la variable N$ se le asigna la constante numérica 25.

```
100 DATA GIL, LOPEZ, PEREZ
120 DATA A$, B$, C$
```

A las variables A$, B$ y C$ se le asignan los valores «GIL»; «LOPEZ»; y «PEREZ». No es preciso poner las constantes entre paréntesis.

Caso b:

Realice el siguiente programa:

```
100 READ A$, B$, C$
120 PRINT A$; B$; C$
140 DATA ESTO ES, UNA, PRUEBA

RUN

ESTO ESUNAPRUEBA
```

Modifique el programa de la siguiente forma:

```
140 DATA  ESTO ES , UNA , PRUEBA

RUN

ESTO ESUNAPRUEBA
```

El resultado permanece inalterable a pesar de haber dejado un espacio en blanco detrás de cada palabra. Realice ahora el programa así:

```
100 READ A$, B$, C$
120 PRINT A$; B$; C$
140 DATA "ESTO ES", " UNA ", "PRUEBA"

RUN

ESTO ES UNA PRUEBA
```

En la mayoría de las computadoras se requiere encerrar entre comillas las cadenas cuando le precede o sigue un espacio en blanco, coma o dos puntos.

```
100 READ A$
120 PRINT A$
140 DATA "MIGUEL, GARCIA"

RUN
MIGUEL, GARCIA
```

3.5. Marca o constante fin de fichero

Si las sentencias DATA contienen datos permanentes y se desea un número determinado de ellos es conveniente, en ocasiones, un indicador *fin de fichero* o *fin de lectura,* con lo que se interrumpirá la lectura y se evitarán mensajes de error OUT OF DATA. El indicador suele ser un número o palabra concreta que se define previamente.

Realice el programa siguiente que calcula la media de los valores contenidos en las sentencias DATA:

```
10 READ X
20 S=S+X
30 N=N+1
40 GOTO 10
50 PRINT "Media ="; S/N
60 DATA 10, 20, 30, 40, 50
100 END

RUN
OUT OF DATA IN 10
```

El programa ha leído todos los datos de la línea 60, pero como la línea 40 transfiere el control siempre a la línea 10, a la sexta iteración nos faltan datos y se obtiene un mensaje de error.

Ponemos en la línea 60 un indicador de fin de lectura, por ejemplo −1, y añadimos también una nueva línea.

```
15 IF X=-1 THEN 50
```

El progama quedaría así:

```
10 READ X
15 IF X=-1 THEN 50
20 S=S+X
30 N=N+1
40 GOTO 10
50 PRINT "Media ="; S/N
60 DATA 10, 20, 30, 40, 50, -1
100 END

RUN
Media = 30
```

Al llegar la lectura de la sexta iteración se asigna el valor −1 a X, por lo que el control del programa se transfiere a la línea 50, con lo que se imprime el resultado buscado y se evita el error OUT OF DATA.

4. SENTENCIA RESTORE

El programa realiza la lectura de los datos almacenados en las sentencias DATA mediante *«un puntero de datos»* interno que apunta siempre a la próxima constante a leer, por lo que al leer el último dato ya no se podrán leer más datos dentro de un mismo programa. Sin embargo, en numerosos programas se necesitan leer varias veces todos o parte de los datos almacenados en sentencias DATA. Para conseguir esta(s) nueva(s) lectura(s) se utiliza la sentencia RESTORE.

Formato:

RESTORE

La ejecución de la sentencia DATA posiciona el puntero interno de datos en la primera constante de la primera línea DATA, por lo que se habilita a la computadora para usar datos almacenados en sentencias DATA más de una vez.

En otras palabras, tras la ejecución de la sentencia RESTORE, la siguiente sentencia READ ejecutada comienza la lectura del primer dato de la primera sentencia DATA.

La sentencia RESTORE *permite reutilizar las sentencias* DATA.

Ejemplo:

```
100 DATA 1, 2, 4, 6, 7, 8, 9, 12, 25
120 READ A, B, C, D
140 PRINT A; B; C; D
160 READ E, F, G
180 PRINT E; F; G
200 RESTORE
220 READ H, J
240 PRINT H; J
260 READ L, M, N, P, Q
280 PRINT L; M; N; P; Q
300 END

RUN
 1  2  4  6
 7  8  9
 1  2
 4  6  7  8  9
```

Las variables H, J, L, M, N, P, Q han tomado los siete valores iniciales de la sentencia DATA debido a la sentencia RESTORE.

El puntero de DATOS se reposiciona al comienzo de la línea de más baja numeración DATA tras la ejecución de una sentencia RESTORE.

4.1. Sentencia RESTORE ampliada

Algunas computadoras permiten una variación de la sentencia RESTORE (por ejemplo, Amstrad, MSX, HP-85, HP-87, HP-9845, IBM/PC, ZX Spectrum, etc.) en la que se expresa el número de línea sobre la que se posicionará el puntero de DATOS, con lo cual aumentará la potencia de RESTORE al poder comenzar la relectura en una línea DATA diferente de la primera.

Formato: | *n* RESTORE *m* |

n Número de línea.
m Número de línea de la sentencia DATA en la que se quiere posicionar el puntero de datos.

Si se incluye el número de línea en la sentencia RESTORE, se accede al primer elemento de la sentencia DATA especificada.

Ejemplo:

```
10 READ A, B, C
20 READ E, F, H
30 PRINT A; B; C; E; F; H
40 RESTORE 120
50 READ X, Y, Z
60 PRINT X; Y; Z
100 DATA 1, 2, 3
120 DATA 6, 7, 8
199 END

RUN
 1  2  3  6  7  8
 6  7  8
```

Al ser RESTORE 120 en vez de RESTORE, los datos que se lean y asignen en la línea 50 son los correspondientes a la línea 120.

4.2. Programa de aplicación completo

Consideremos nuevamente el programa del apartado 3.1 que nos permitirá calcular las notas medias de una clase de una determinada asignatura.

Si ahora quisiéramos que este programa realizase las medias de otras asignaturas, no habría más remedio que volver a ejecutar de nuevo el programa, ya que las sentencias DATA que contienen los nombres (apellidos en nuestro caso) han sido ya leídas.

Mediante la sentencia RESTORE se podrían realizar las notas medias de diferentes asignaturas, puesto que la misma nos permitiría la lectura repetida de los datos.

Supondremos igual que en el apartado 4.1 que en sentencias DATA se han introducido nombres y número de alumnos. Por lo que cada vez que un alumno sea ALTA o BAJA habrá que añadirlo o eliminarlo de dichas líneas DATA.

Por teclados se introducirán las diferentes asignaturas, con un bucle anidado K que se conecte a los I, J ya existentes.

El programa modificado podría ser:

```
100 REM Notas medias
105 INPUT "Numero de asignaturas"; H
110 FOR I=1 TO H
115 INPUT "Nombre de la asignatura"; B$
120 READ N
140 FOR J=1 TO N
160 READ A$
180 PRINT "Cuantas notas del alumno "; A$
200 INPUT C
210 FOR K=1 TO C
```

```
220 PRINT "Nota numero";K
240 INPUT A
260 S=S+A
280 NEXT K
300 M=S/C
310 PRINT "Asignatura ";B$
320 PRINT "Media alumno ";A$; " ="; M
330 S=0
340 NEXT J
360 RESTORE
380 NEXT I
400 DATA 12
420 DATA "Mortimer", "McKoy", "Flanagan", "Lopez"
440 DATA "Perez", "Garcia", "Ruiz", "Martinez", "Gil",
    "Gomez", "Gila"
460 DATA "Marin"
```

El programa, como se ve, es igual al del párrafo 3.1, al que se han añadido las líneas 105, 110, 115, 310, 340 y 360. Al ejecutarse el programa, se introduce el número de asignaturas a definir, y tras introducir el nombre de la 1.ª asignatura, se efectúan las medias de los 12 alumnos en dicha asignatura. Al llegar el programa a la línea 360 RESTORE, el puntero de DATOS se posicionará de nuevo en la línea 400. La línea 380 NEXT I hace que el valor de I siguiente sea 2, con lo que se retorna a la línea 110 y se introduce el nombre de la 2.ª asignatura y a partir de este punto se repite completamente el programa.

5. ANCHURAS DE LA PRESENTACION

Actualmente son muchas las computadoras que le permiten cambiar la anchura de su presentación o lo que es igual, el número de columnas que se pueden visualizar en pantalla o en impresora. Algunas de estas computadoras son IBM/PC y compatibles, Amstrad y MSX. Veamos cuáles son las sentencias utilizadas en cada caso.

Presentación en pantalla

IBM.
Dos tamaños o anchuras de pantalla: 40 y 80 columnas. La sentencia/orden que conmuten ambas presentaciones es WIDTH y su formato:

WIDTH 80: Activación pantalla de 80 columnas (desactiva pantalla de 40 columnas).
WIDTH 40: Activación pantalla de 40 columnas (desactiva pantalla de 80 columnas).

Amstrad.
La orden/sentencia MODE determina el número de caracteres por línea.
MODE 0: 20 caracteres por línea
MODE 1: 40 caracteres por línea
MODE 2: 80 caracteres por línea

MSX.
La sentencia WIDTH establece la anchura de la pantalla. Los formatos son:

WIDTH ⟨ *expresión numérica*>

Los modos de pantalla son dos: 0 y 1.

Pantalla 0: El valor por defecto es 37 columnas y el valor máximo 40.
Pantalla 1: El valor por defecto es 29 y el valor máximo 32.
WIDTH 30 anchura máxime de pantalla, 30 columnas.
WIDTH 30 anchura máxima de pantalla, 20 columnas.

NOTA: *En el caso de las computdoras MSX e IBM/PC, la sentencia SCREEN que establece las pantallas 0 y 1 está íntimamente relcionada con WIDTH que establece las anchuras de presentación.*

Presentación en impresora

Si utiliza impresoras, se puede establecer la anchura de impresión mediante la sentencia WIDTH LPRINT, que actúa de igual forma que WIDTH, con la variante de permitir anchuras entre 15 y 255, siendo las columnas usuales 80 y 132.

6. INSTRUCCIONES DE PRESENTACION DE RESULTADOS

En los capítulos 1 y 2 se vio la sentencia PRINT con sus diversas variantes, que permitían la representación de mensajes o datos de cualquier tipo sobre la pantalla, pero con unas limitaciones impuestas por los separadores (,) (;) o espacio en blanco.

Estas limitaciones hacen que el *formateado* (proceso de disposición u organización de la información en una pantalla o impresora) con sólo la sentencia PRINT se puede convertir en una tarea compleja y penosa, si se desean obtener formatos de salidas agradables a la vista. Existen tres funciones que facilitan considerablemente la presentación de datos en pantalla/impresora: SPC, TAB y POS.

La edición de resultados se potencia con una variante de la sentencia PRINT que en la actualidad incluyen la mayoría de las versiones de BASIC y que se conoce como la sentencia PRINT USING.

Dado que la presentación de resultados parciales o definitivos, mensajes orientativos, etcétera, forma parte inseparable de un programa, le aconsejamos que estudie con detenimiento este apartado ya que forzosamente, además de la sentencia PRINT, tendrá que incluir alguna de las funciones y sentencias mencionadas anteriormente en cualquier programa.

Además de las funciones/sentencias ya mencionadas se verán también las sentencias LOCATE, PRINT AT y PRINT @, conocidas como PRINT *retícula,* que permiten la impresión directa en cualquier posición de la pantalla. Esta sentencia existe en alguns versiones de BASIC, esencialmente basadas en MICROSOFT, COMO IBM PC, MSX, Amstrad, TRS-80, DRAGON, etc. Esta sentencia equivale a PRINT AT que utilizan las computadoras SINCLAIR, ZX Spectrum.

6.1. Función SPC

SPC, abreviatura de la palabra inglesa «SPACE» (espacio), es una función de espaciado y permite desplazar el cursor a la derecha un número especificado de posiciones o, lo que es igual, insertar un número específico (n) de espacios.

Formato: *SPC (exp)*

exp Variable o expresión numérica.

Por ejemplo, PRINT SPC (20); "HOLA" imprime 20 espacios en blanco seguidos por la palabra HOLA.

El argumento de SPC será el número de espacios en blanco que se desea insertar o espaciar. El cursor se desplazará a la derecha el número de espacios indicado por el argumento, borrándose cualesquiera caracteres que sobrepase.

La función SPC desplaza el cursor a la derecha desde cualquier posición de columna de la pantalla (impresora) en que pueda estar el cursor cuando se encuentre la función SPC.

NOTAS:

1. Es necesario un punto y coma después de la palabra SPC, ya que si fuera una coma comenzaría la visualización del texto diez caracteres (espacios en blanco) después de la terminación del número de espacios indicados por SPC.
2. Se puede incluir la función SPC después de una sentencia PRINT que origina simplemente que se visualice/imprima el siguiente carácter el número de posiciones después de la posición actual del cursor que indica el argumento.

Programa ejemplo:

```
10 PRINT "AAA"; SPC(5); "BBB"

RUN
AAA     BBB
```

VARIANTES DE SPC

La función SPC es muy similar a la función SPACE$, ya estudiada. De hecho puede utilizar SPACE$ (5) en lugar de SPC en el ejemplo anterior, pero hay una diferencia importante: SPC sólo se puede utilizar dentro de una sentencia importante: SPC sólo se puede utilizar dentro de una sentencia PRINT o LPRINT. La razón para esta limitación es que al contrario de SPACE$, SPC no devuelve una cadena y sólo puede realizar su tarea como parte de PRINT o LPRINT.

La función SPC toma también los nombres SPA y SPACE en algunos modelos de computadoras.

6.2. Función TAB

La función TAB se utiliza, junto con PRINT, de un modo similar a la tecla TAB (de tabulación) de una máquina de escribir. Esta función desplaza el cursor a la derecha a la posición de columna especificada, imprimiendo a partir de esa columna.



exp Variable o expresión numérica que indica la columna donde se comienza a imprimir o visualizar.

Cuando la sentencia PRINT está seguida por TAB(n), la computadora inserta un número de espacios en blanco *n* antes de que se imprima el mensaje correspondiente.

El argumento *(exp)* de TAB debe ser siempre positivo y debe ser menos que el número de espacios permitidos por línea.

El cursor no se desplaza si el argumento no está a la derecha de la posición actual. TAB da lugar a la impresión de espacios en blanco a medida que se desplaza el cursor a la derecha, con lo que se borra todo lo que hubiera en la pantalla con anterioridad.

El argumento de TAB debe estar comprendido entre 1 y 255. Si este argumento es mayor que la anchura normal del dispositivo de salida (40 y 80 en pantalla, 132 en impresora), el cursor se desplaza hacia abajo en una línea y reanuda la cuenta en el margen izquierdo. Si el valor del argumento es 0, TAB desplaza el cursor a la columna 256. Si el valor del argumento está fuera del intervalo 0-255, se obtiene un mensaje similar a ?ILLEGAL QUANTITY ERROR.

Ejemplos:

```
PRINT TAB (5);
PRINT TAB (X);
PRINT TAB (3*X + 2*Y)
```

NOTAS:

1. TAB debe estar seguido por una coma o un punto y coma, según los intérpretes de BASIC.
2. Si se utiliza más de una función TAB en una línea, los números de líneas deben aumentar progresivamente.

Programa ejemplo:

```
10 REM Funcion "TAB"
20 PRINT TAB (6); "TAB 6"
30 X=12
40 PRINT TAB (X); "TAB 12"
50 PRINT TAB (18); "TAB 18"

RUN

      TAB 6

            TAB 12

                  TAB 18
```

6.3. Función POS

La función POS proporciona la posición actual horizontal (columna) del cursor.

Formato: POS *(exp)*

exp Variable o expresión numérica.

La posición se devuelve como un número igual al número de la columna en la que está posicionado el cursor. El argumento *(exp)* es ficticio (falso) y suele ser 0.

```
PRINT "La posicion del cursor es"; POS (0)
La posicion del cursor es 25
```

POS proporcionará un valor entre 1 y 40 (pantallas de 40 columnas) o entre 1 y 80 (pantallas de 80 columnas). Las posiciones de los caracteres comienzanen 0 para el carácter más a la izquierda.

6.4. Programas ejemplo de las funciones SPC, POS y TAB

A continuación, para afianzar el funcionamiento de SPC/POS/TAB, vamos a describir diferentes programas ejemplos.

Ejemplos:

```
10 PRINT SPC (10); "*"      Deja diez espacios en blanco
99 END
RUN
          *

10 PRINT SPC (6); "*"; SPC (15); "#"; SPC (10); "/"
99 END
RUN
      *               #          /

10 FOR I=1 TO 9
20 PRINT SPC (I); I
30 NEXT I
99 END
RUN
  1
   2
    3
     4
      5
       6
        7
         8
          9

10 INPUT A, B, C
20 PRINT SPC (5); A; SPC (10); B; SPC (10); C
99 END
RUN
?4, 7, 15
      5           7            15

10 FOR K=1 TO 6
20 PRINT TAB (K); K
30 NEXT K
40 END
RUN
  1
   2
    3
     4
      5
       6

10 PRINT TAB (5); "*"; TAB (7); "+"; TAB (30); "#"
RUN
    *  +                          #

10 PRINT TAB (15); "SR. MARTINEZ"
20 PRINT TAB (18); "CALLE MAYOR 105"
30 PRINT TAB (21); "28005 MADRID"
RUN
               SR. MARTINEZ
                   CALLE MAYOR 105
                      28005 MADRID

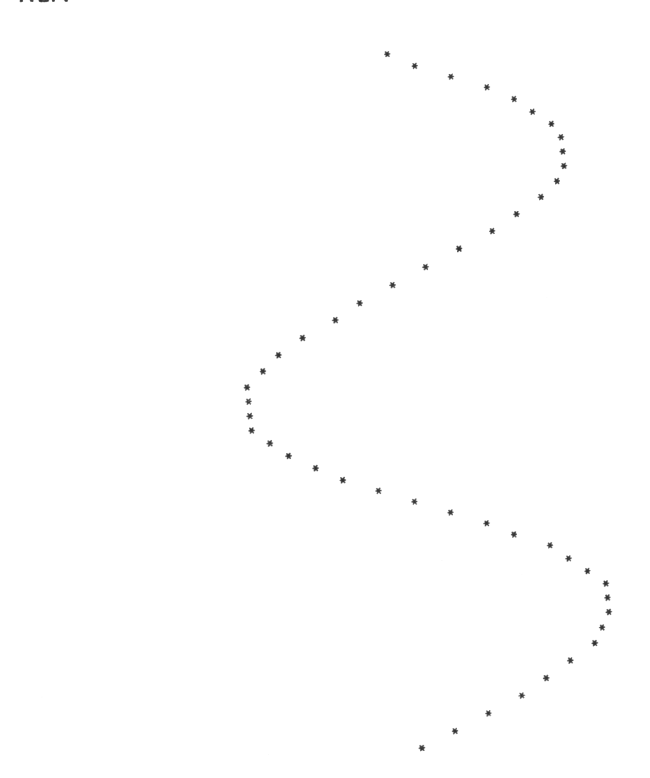
100 IF POS (0)>30 THEN PRINT "FIN"
```

Cuando la posición del cursor esté después de la columna 30 se imprimirá FIN.

Programa 1: Dibujo de la función Y=SIN(X).

```
10 FOR X=0 TO 4 * 3.14159   STEP .2
20 Y=SIN (X)
30 Y=20*Y+20
40 PRINT TAB(Y); "*"
50 NEXT X

RUN
```



Programa 2:

```
10 FOR I=1 TO 6
20 PRINT TAB (I); "*"
30 NEXT K
99 END

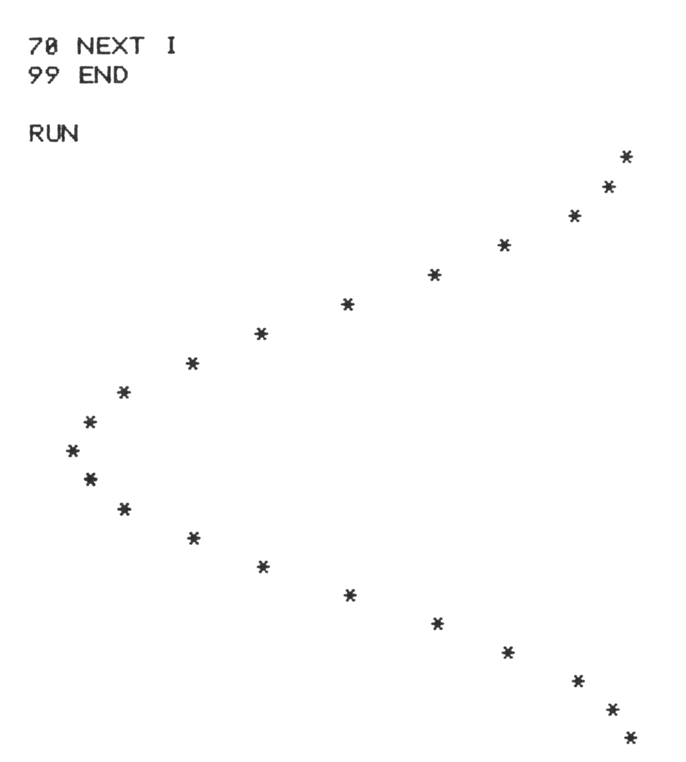
RUN
 *
  *
   *
    *
     *
      *
```

Programa 3: Trazado de la curva coseno de X (*Función* TAB).

```
10 X=0
20 FOR I=1 TO 21
30 Y=COS (X)
40 H=Y*16+18+.5
50 PRINT TAB(H); "*"
60 X=X+3.141592/10
```

```
70 NEXT I
99 END

RUN
```



6.5. Modo de presentación de texto

La pantalla de una computadora se puede utilizar de dos diferentes modos para visualizar salidas de programa claras y concisas. Estos dos modos de visualización son: Texto y Gráficos. El modo Texto permite poner cualquiera de los 256 caracteres y símbolos en la pantalla. El modo Gráficos permite dibujar figuras controlando puntos individuales en la pantalla. En este párrafo analizaremos el modo Textos. El modo Gráficos no se estudia en este libro.

El modo Texto se puede utilizar en cualquier micro. El procedimiento básico para la creación de la salida de visualización es inicializar la pantalla, mover el cursor donde lo desee, y a continuación *poner los datos en la pantalla* con PRINT, PRINT USING o alguna de sus variantes.

INICIALIZACION EN PANTALLA

Antes de situar información en la pantalla deberá inicializarse activando la anchura y limpiando la misma. Esta operación se realiza con dos órdenes BASIC ya analizadas: WIDTH y CLS.

Recordemos los formatos de estas órdenes:

`WIDTH 40`	Anchura de pantalla a 40 columnas.
`WIDTH 80`	Anchura de pantalla a 80 columnas.
`CLS`	Limpieza de la pantalla.

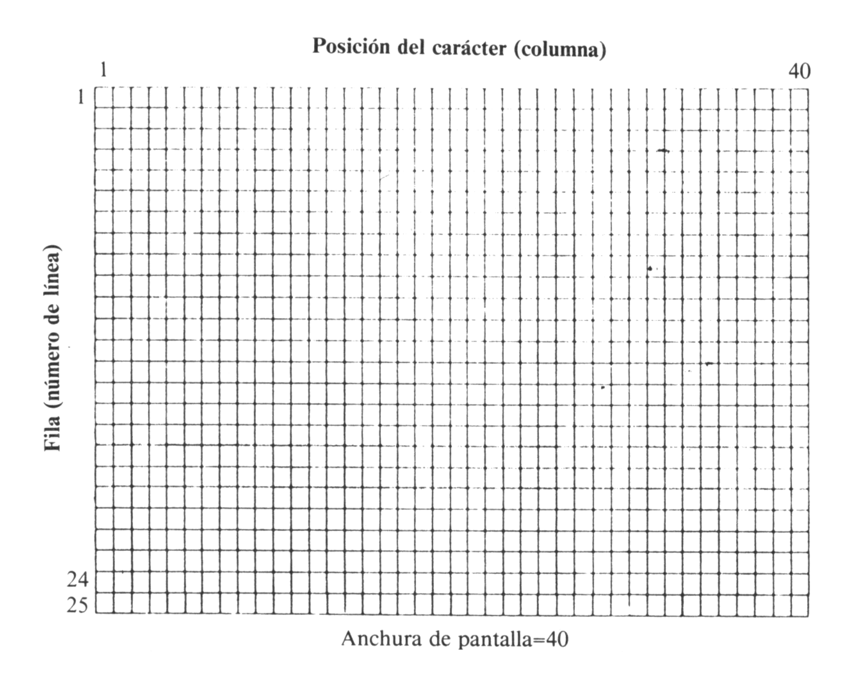
6.5.1. Posicionamiento del cursor (sentencia LOCATE)

El control de la posición del cursor puede crear salidas con formatos definidos, tales como tablas, gráficos y diagramas.

La posición del cursor se define por su número de línea y la posición del carácter dentro de esa línea. La sentencia LOCATE sirve para imprimir cualquier cosa que desee en la pantalla.

En primer lugar, necesita tener un medio de asignación de una posición en la pantalla: esto se consigue por medio de dos números: número de fila, que define la posición verticalmente,

y el número de columnas que define la posición horizontalmente. La fila puede ser cualquier número entero entre 1 y 25 mientras que las columnas puede ser cualquier número entre 1 y 40 o 1 y 80 según la anchura de la pantalla seleccionada. Las coordenadas del cursor se muestran en la Figura 7.1.



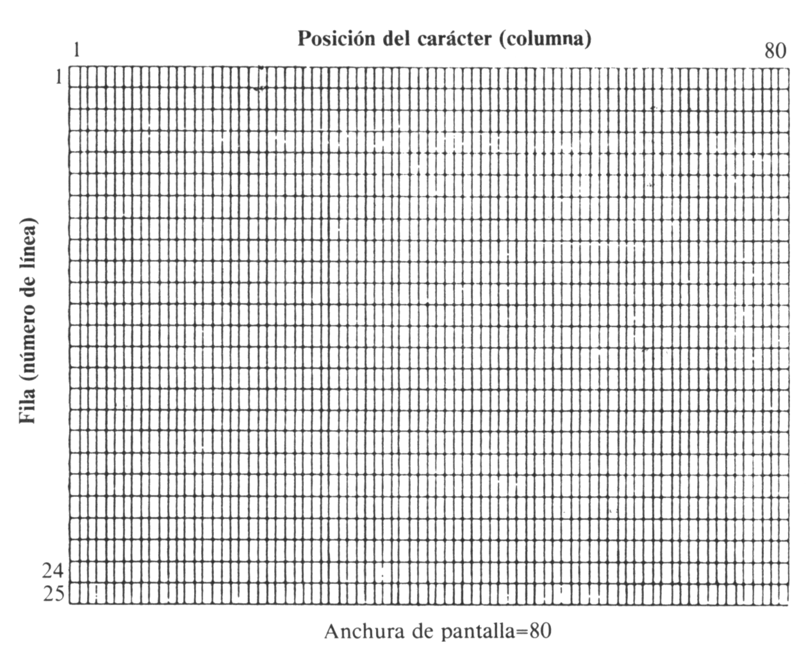


Figura 7.1. Mapas de coordenadas de caracteres en pantalla de texto.

El cursor se puede situar en cualquier parte de la pantalla con la sentencia LOCATE.

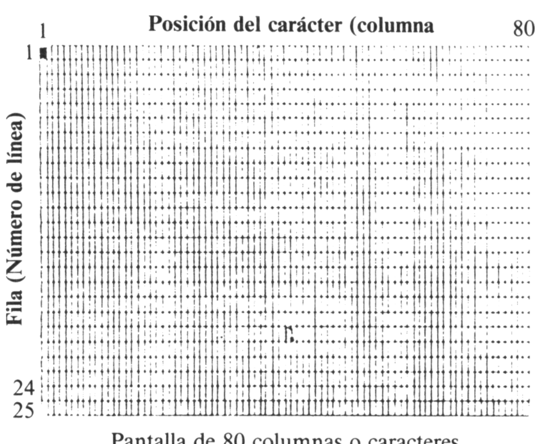
Formato: | *n* LOCATE *fila, columna, cursor, comienzo, parada* |

n	Número de línea.
fila	Expresión numérica en el rango 1 a 25.
columna	Expresión numérica en el rango 1 a 40 (para WIDTH=40) o 1 a 80 (para WIDTH=80).
cursor	0, del cursor no es visible. 1, el cursor es visible.
comienzo	Línea donde comienza el barrido del cursor; expresión numérica en el rango 0 a 31.
parada	Línea donde termina el barrido del cursor; expresión numérica en el rango 0 a 31.

Cuando se está ejecutando un programa el cursor está desconectado normalmente. Utilice LOCATE, 1 para conectarlo.

Algunos ejemplos de aplicación son:

`10 LOCATE 1,1`	Posiciona el cursor en la esquina superior izquierda de la pantalla.
`40 LOCATE ,,1`	Pone intermitente el cursor invisible, su posición permanece invariable.
`100 LOCATE 4,3,1,0,7`	Llevar el cursor hasta la línea 4, columna 3, hace visible el cursor, comienza el barrido en la línea 0 y termina en la línea 7 (ocupa toda la celda del carácter).
`10 LOCATE 20,40,1,0,13`	Mueve el cursor a la línea 20, columna 40 y ocupa toda la celda completa.



Pantalla de 80 columnas o caracteres

Si se omite algún parámetro, BASIC utiliza los valores definidos últimamente. La sentencia LOCATE se utiliza en unión con la setencia PRINT. En primer lugar la computadora IBM/PC comienza indicando la posición y a continuación dice lo que se ha de imprimir. Veamos un programa ejemplo.

```
5 REM Demo 1
10 CLS
20 LOCATE 1.1
30 PRINT "esquina superior iz."
40 LOCATE 24,1
50 PRINT "esquina inferior iz."
60 LOCATE 24,60
70 PRINT "esquina inferior de."
80 LOCATE 1,60
90 PRINT "esquina superior de."
99 END
```

```
esquina superior iz.                          esquina superior de.




esquina inferior iz.                          esquina inferior de.
```

Figura 7.2. Salida del programa Demo 1.

En las computadoras IBM/PC y compatibles, la línea 25 o fila 25 (inferior) de la pantalla está reservada para las teclas de función programables. Mediante las sentencias KEY ON y KEY OFF se puede presentar o borrar dicha línea inferior de la pantalla.

KEY OFF Borra los mensajes de las teclas de función programables en la línea inferior de la pantalla.

KEY ON Presenta de nuevo la línea de teclas de función programables.

Para probar la visualización, ejecute la orden

```
KEY OFF
```

y a continuación el programa siguiente:

```
10 LOCATE 25,30
20 PRINT "Hola Mackoy"
30 LOCATE 15,1
99 END
```

Como curiosidad, trate de listar el progrma con LIST y observará que la línea 25 no desaparece de la pantalla. El enrollamiento típico de la pantalla se produce en las 24 primeras líneas. Si desea borrar toda la pantalla, incluyendo la línea 25, lo deberá hacer con la orden CLS.

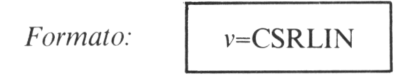
6.5.2. Posición del cursor [*sentencias* CSRLIN y POS(0)]

La posición actual del cursor en la pantalla se puede obtener utilizando las funciones CSRLIN y POS(0), en computadoras IBM/PC y compatibles, así como en las computadoras MSX.

CSRLIN

Devuelve la coordenada vertical de la posición del cursor.

Formato:



v=CSRLIN

v Variable numérica.

POS

Devuelve la coordenada horizontal de la posición del cursor.

Formato:



v=POS(0)

v Variable numérica.

Ejemplo:

```
10 X=POS(0):Y=CSRLIN
20 LOCATE 20,1
30 PRINT "HOLA MACKOY"
40 LOCATE Y,X
```

La posición del cursor puede ser útil; por ejemplo, cuando el cursor se debe mover y a continuación retornar a su posición original.

6.5.3. *Sentencia* PRINT

Este sentencia es específica de algunas computadoras, como el TRS-80 y los modelos DRAGON-32/64/200 y específica exactamente donde debe empezar la impresión.

Formato:



n PRINT @ *exp, l*

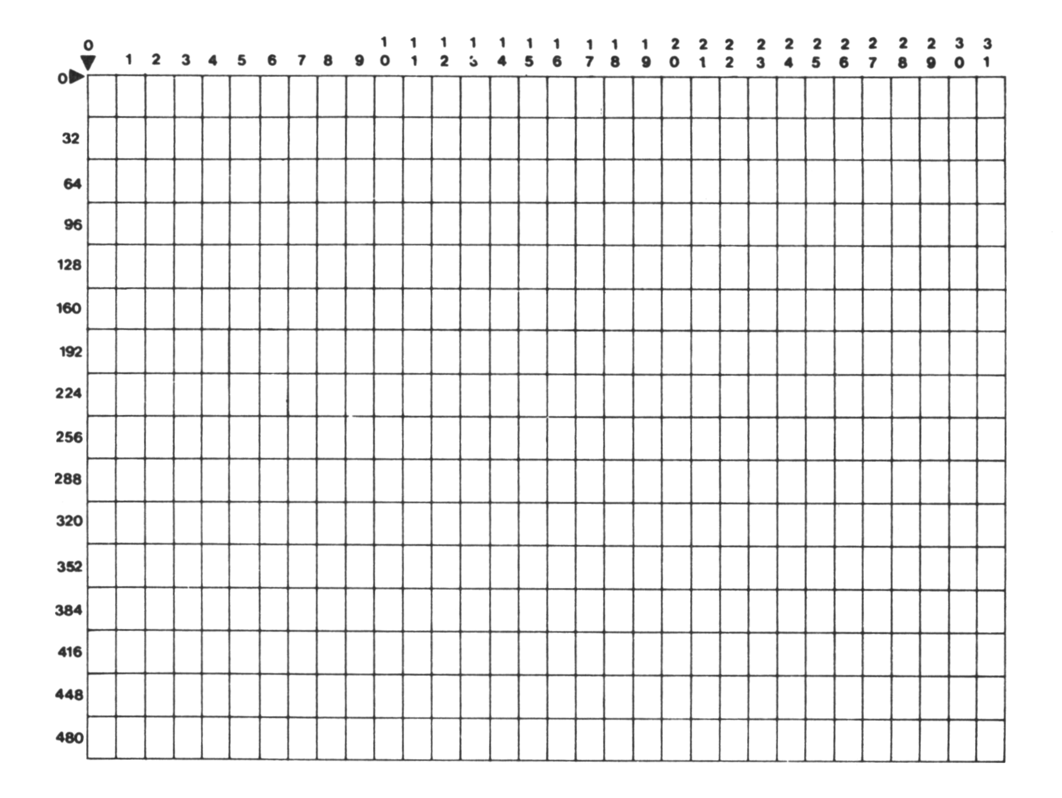
n Número de línea.
exp Variables o expresión numérica que indica la posición exacta de la pantalla.
l Lista de elementos, igual a la utilizada en cualquier sentencia PRINT.

Dado que la localización exacta en la pantalla diferirá de su formato, veremos dos ejemplos representativos con la TRS-80 y la DRAGON-32.

DRAGON-32/64/200

La pantalla se considerará dividida en 512 posiciones, cada una de las cuales puede albergar un carácter. 512 viene del producto 16×32, ya que su pantalla representa 16 filas de 32

caracteres. Para realizar una determinada posición, la pantalla se trata como si fuera una retícula o cuadriculado (Figura 7.3).



PRINT @ *expresión, lista de elementos*

La expresión puede ser un número, una variable o una expresión aritmética cuyo valor esté comprendido entre 0 y 511.

```
PRINT @ 32*(LINEA-1) , A
```

imprimirá el valor de A al comienzo de una determinada línea de la pantalla, dependiendo del valor de la variable LINEA entre 1 y 16.

NOTAS:

1. Siempre que se usa PRINT @, en la línea última de la pantalla se produce un avance de línea automático que produce un movimiento de una línea hacia arriba en toda la pantalla. Para suprimir este efecto perjudicial debe utilizarse un punto y coma al final de la sentencia correspondiente.

```
100 PRINT @120, "MCKOY";
100 PRINT @200,200;:PRINT @250, 250
```

2. Al utilizar la sentencia PRINT @ se suele añadir al final un bucle sin fin para evitar que aparezca el mensaje OK (listo, preparado) que aparece cuando termina el programa.

```
100 PRINT @ 400, 400;
110 GOTO 110
```

TRS-80

La pantalla de la computadora TRS-80 es un retículo de 1024 cuadrados o celdillas, numerados desde 0 a 1023, o lo que es igual 16 × 64 (filas × columnas).

```
100 PRINT @420, "HOLA"
120 PRINT @820, "HOLA"
```

Ambas líneas imprimen HOLA en la pantalla, la 100 comenzando en la posición 420 y la 120 en la posición 820.

PRINT @ expresión, lista de elementos

La expresión puede ser un número comprendido entre 0 y 1023.

```
PRINT @Y*64+X, "MENSAJE"
```

Esta sentencia posiciona el cursor en las coordenadas X, Y (número de fila, número de columna) de la pantalla y visualiza el MENSAJE entre comillas.

```
PRINT @5*64+10, "DEMO";
```

imprime DEMO en la quinta fila, columna décima.

```
        0 1 2 3 4 5 6 7 8 9 10
Fila 0
Fila 1
Fila 2
Fila 3
Fila 4
Fila 5            DEMO
```

La siguiente línea es equivalente a la anterior.

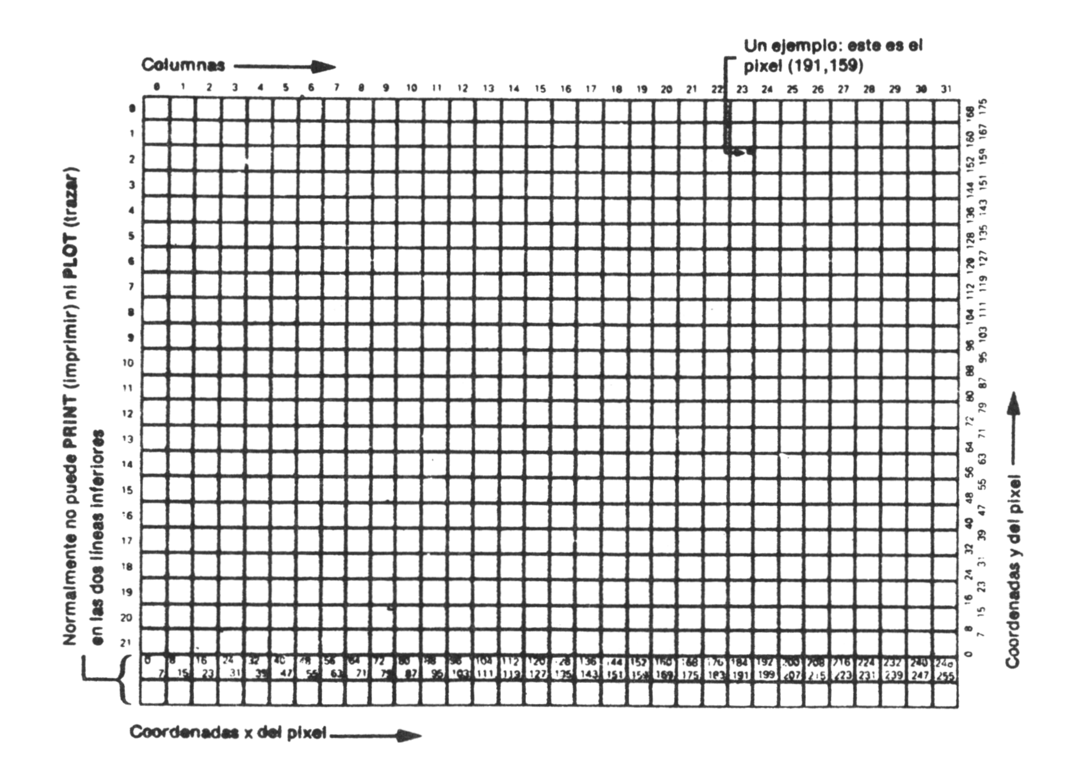
```
PRINT @330, "DEMO";
```

Las notas 1 y 2 relativas al DRAGON-32 son de aplicación también aquí.

6.5.4. Sentencia PRINT AT

Las computadoras ZX Spectrum de la firma SINCLAIR tienen una sentencia específica de gran potencia para impresión, PRINT AT. Es similar a PRINT @, pero mucho más práctica, ya que la posición inicial de la impresión es más fácil de expresar.

La pantalla de la computadora ZX Spectrum posee 24 filas y 32 columnas, de las que dos filas se reservan para comunicación usuario/computadora. La pantalla sería equivalente a un *reticulado* como el siguiente:



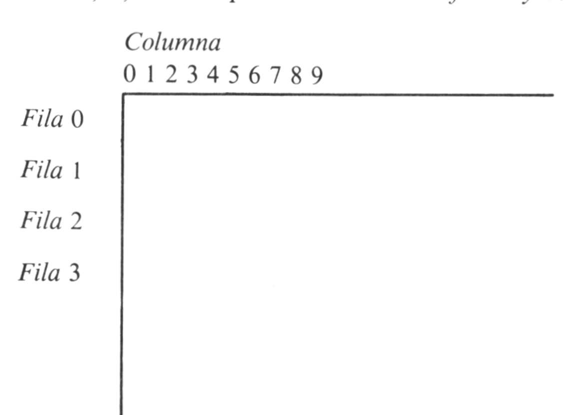
Cada posición del reticulado está perfectamente definida por la intersección de una fila y una columna, o sea unas coordenadas (*x, y*), donde *x* es la fila e *y* la columna.

Formato:

n PRINT AT x, y; *"mensaje o variables válidas"*

n	Número de línea.
mensajes, variables	Mensaje o variables válidas.

Las filas y columnas comienzan su numeración en 0, o sea (0, 1, 2, ...31) para las filas, (0, 1, 2, ... 21), para las columnas.
PRINT AT 3, 7; «X» *Impresión de X en la fila 3 y columna 7*



Además de las sentencias tratadas en este párrafo existe una sentencia que produce una gran flexibilidad a los formatos de cadenas de caracteres o números. Esta sentencia es PRINT USING para presentaciones en pantallas, que se transforma en LPRINT USING para impresoras.

7. SENTENCIA PRINT USING

PRINT USING se utiliza por muchas computadoras (principalmente basadas en MICROSOFT, IBM PC, MSX, TRS-80, Amstrad, DRAGON-32, HP, etc.) como una sentencia PRINT que permite imprimir números o cadenas *utilizando* (USING) un formato específico, y por ello es la sentencia PRINT más potente disponible en BASIC.

La función de la sentencia PRINT USING es imprimir, además de caracteres o números, expresiones que utilizan un formato determinado.

Formato: PRINT USING *formato; lista de expresiones*

El *formato* es el conjunto de caracteres que representan el campo y el formato de las cadenas de caracteres o números que se han de imprimir.

Lista de expresiones son las utilizadas en sentencias PRINT: cadenas o expresiones numéricas que se imprimen; separadas por comas o puntos y comas.

La sentencia PRINT encuadra (ajusta) las cadenas y números por la izquierda.

```
10 A=125
20 B=13456
30 PRINT A
40 PRINT B

RUN
 125
 13456
```

Sin embargo, el tratamiento de los números, sobre todo en gestión y cálculos científicos, se hace normalmente encuadrándolos por la derecha.

```
  125
13456
```

La sentencia PRINT USING permite encuadrarlos por la derecha.

```
10 A=125: B=13456
20 PRINT USING "######"; A
30 PRINT USING "######"; B

RUN
   125
 13456
```

La serie de caracteres # representa el formato de edición. En este ejemplo se pueden editar números de hasta cinco cifras o dígitos.

Ejemplo:

```
PRINT USING "##.##"; M
PRINT USING "$$"; D
```

La expresión de cadena consiste en campos de datos que describen el formato usado para la salida de la lista de expresiones y pueden tener caracteres adicionales (espacios en blanco, p.p.). La separación de campos se hace mediante *separadores* o *especificadores de campo* tales como #, *, $, etc.

7.1. Edición de zonas «caracteres»: campos de cadenas

Cuando PRINT USING se utiliza para imprimir cadenas, uno de los caracteres de formateado siguientes se pueden utilizar para obtener el formato del campo de cadena:

¡ Indica que sólo el primer carácter de la cadena se debe imprimir.

n espacios\\ Indica que n+2 caracteres de cada cadena se deben editar. Si las barras inclinadas se introducen sin espacios, se imprimirán dos caracteres.
Si hay un espacio se imprimirán tres caracteres y así sucesivamente.
Si la cadena es más larga que el campo (formato) se ignoran los caracteres suplementarios.
Si el formato es más largo que la cadena, ésta se ajustará por la izquierda y se completará con espacios en blanco a la derecha.

```
10 A$="MORTIMER"
20 B$="FLANAGAN"
30 PRINT USING "!"; A$; B$
40 PRINT USING "\  \"; A$; B$
50 PRINT USING "\   \"; A$; B$

RUN
MF                M primer carácter de MORTIMER, F primer carácter de FLANAGAN
MORTFLAN          4 primeros caracteres de MORTIMER y FLANAGAN
MORTIFLANA        5 primeros caracteres de MORTIMER y FLANAGAN
```

& Permite la utilización de cadenas de longitud variable. En este caso la cadena se imprime exactamente tal como se ha introducido.

```
10 A$="MORTIMER"
20 B$="FLANAGAN"
30 PRINT USING "!"; A$;      Se imprime, M
40 PRINT B$                  Imprime B$

RUN
MFLANAGAN
```

7.2. Edición de zonas «numéricas»: campos numéricos

Cuando PRINT USING se utiliza para imprimir números, se pueden utilizar los siguientes caracteres especiales para formatear el campo numérico:

El signo se utiliza para representar cada posición de un dígito. Todas las posiciones se ocuparán, el número se alinea por la derecha y se completa con blancos a la izquierda. El número de signos usados establece el campo numérico.

```
PRINT USING "#####"; "12345"
12345
```

```
PRINT USING "#####"; 123
  123
PRINT USING "###"; 99.9
100
PRINT USING "#######"; 123.45
    123
PRINT USING "##########"; 1234.57
      1235
```

Los resultados se ajustan por la derecha con las siguientes reglas:

- Las partes decimales se redondean por exceso.
- Si la cantidad es negativa, un signo (–) se sitúa delante de la primera cifra significativa. Si se puede presentar, deberá preverse un formato suficientemente grande.
- Si el formato es demasiado pequeño se imprime, pero:
  — precidida del signo %
  — no ajustado con los otros números.

```
PRINT USING "##"; 123
%123
```

. Un número decimal se puede insertar en cualquier posición del campo.
Si el formato precisa que una cifra debe aparecer delante del punto decimal, se imprimirá siempre una cifra, incluso cuando sea el cero.

```
PRINT USING "##.##"; 98.654
98.65
PRINT USING "###.##"; 13.4695
 13.47
PRINT USING "###.##"; 1492.878
%1492.88
PRINT USING "###.##"; 146
146.00
PRINT USING "###"; 18.76
 19
PRINT USING "##.##"; 10.2, 5.2, 66.789, .234
10.20    5.20    66.79     0.23
```

\+ Un signo + al principio o al final del formato o campo numérico provocará la impresión del signo + o – antes o después del número.

```
PRINT USING "+#####"; 123
  +123
PRINT USING "+#####"; -123456
%-123456
PRINT USING "###.##+"; 123.45
123.45+
PRINT USING "###.##+"; -16
 16.00-
```

– Un signo – al final del formato provocará que los números negativos se impriman con un signo menos detrás del número y los números positivos con un espacio en blanco.

```
PRINT USING "##.##-"; -78.95, 3.40, 75.6, -.91
78.95- 3.40 75.60  0.91-
PRINT USING "###.##-";-27.86
 27.86-
PRINT USING "###.##-"; -1.6
  1.60-
```

** Un doble asterisco al principio del formato provoca el rellenado de la parte no utilizada con asteriscos. Estos ** pueden, en caso de necesidad, ser utilizados para poner cifras. Eso significa que para la zona de impresión existen dos posiciones más que las señaladas mediante #.

```
PRINT USING "**###.#"; 22.49, -0.8, 764.3
***22.5***-0.8**764.3
PRINT USING "**###.##"; 1.492
****1.49
PRINT USING "**######"; -456
****-456
PRINT USING "**###.##"; -575
*-575.00
```

$$ El signo $ (dólar) se utiliza para representar dinero. Un signo de doble dólar al principio del formato provoca la impresión del signo $ a la izquierda de la primera cifra significativa del número. Al igual que la pareja **, el signo $$ ofrece dos posiciones suplementarias para poner en ella cifras, una de las cuales es el signo dólar.
El formato exponencial no se puede utilizar con $$. Con números negativos, el signo a la izquierda se situará delante de $.

```
PRINT USING "$$###.##"; 556.78
$556.78
PRINT USING "$$###.##"; 2.86
   $2.86
PRINT USING "$$###.##"; -765.2
-$765.20
```

**$ El signo **$ al principio de un formato combina los efectos de los dos símbolos anteriores: (**) y ($$). Los espacios no significativos se rellenan con asteriscos y se imprimirá un signo dólar antes que el número. Se dispone de tres cifras suplementarias, una de las cuales es el signo dólar.

```
PRINT USING "**$##.##"; 3.34
***$3.34
PRINT USING "**$#.##"; 1.49
*$1.49
PRINT USING "**$##.##"; 123.45
*$123.45
```

Una coma situada a la izquierda del punto decimal en un formato provoca la inserción de una coma a la izquierda de cada grupo de tres cifras a la izquierda del punto decimal. Una coma situada al final de un formato se imprime como parte de la cadena editada.
La coma no tiene efecto si se utiliza con el formato exponencial (^^^^).

```
PRINT USING "####,.##"; 1234.5
1,234.50
PRINT USING "####.##,"; 1234.5
1234.50,
PRINT USING "########,.##"; 1234567.89
1,234,567.89
```

^^^^ Cuatro acentos circunflejos se pueden situar detrás de un # para precisar que se desea un resultado bajo forma exponencial.
El exponente tendrá la forma E o D+XX.

E para simple precisión.
D para doble precisión.
Los dígitos significativos se ajustan a la izquierda y el exponente se encuadra adecuadamente. A menos que se indique un signo + a la izquierda o un signo — a la derecha, una posición a la izquierda del punto decimal se utiliza un espacio o un signo menos.

```
PRINT USING "##.##^^^^"; 234.56
2.35E+02
PRINT USING ".###^^^^-"; -88888
.889E+05-
PRINT USING "+.##^^^^"; 124
+.12E+04
PRINT USING "##.###^^^^"; 123456
1.2346E+05
```

— Un guión en el formato indica que el carácter que sigue debe ser considerado como una constante carácter.

```
PRINT USING "-!##.##-!"; 12.34
2.34!
PRINT USING "!**###--!"; 1234
1234!
```

El guión puede hacerse un carácter constante situando «—» en la cadena del formato.
Si el número que se va a imprimir es mayor que el campo numérico especificado, un signo de porcentaje se imprime delante del número redondeado.
Si el redondeo produce que el número exceda al campo, el signo porcentaje se imprime delante del número redondeado.

```
PRINT USING "##.##"; 111.33
%111.33
PRINT USING ".##"; .999
%1.00
```

7.3. Programas de aplicación prácticos

El formato de las cadenas que se desean representar puede ser hecho mediante variables de cadenas, etiquetas y cadenas de caracteres.

```
10 AB$="####.##"
20 A=124.624
30 PRINT USING AB$; A

RUN
 124.62
```

Se puede añadir etiquetas o mensajes al formato

```
100 A=123.456
110 PRINT USING "TOTAL IGUAL #####.## PESETAS": A

RUN
TOTAL IGUAL   123.46 PESETAS
```

A continuación se indican varios programas tipo que permiten utilizar adecuadamente la sentencia PRINT USING.

Programa 1: Entrada de datos.

```
10 INPUT A$, A
20 PRINT USING A$; A
30 GOTO 10

RUN
?##.##, 13.13
13.13
?###.##, 13.13
 13.13
?##.##, 131.14
%131.14
?##.##, 13.137
13.14
?+##.##, 14.14
+14.14
?+##.##,-14.14
-14.14
?##.##+, 14.14
14.14+
?##.##+,-14.14
14.14-
?##.##-, 14.14
14.14
?##.##-,-14.14
14.14-
?**##, 14.14
**14
?$$##.##, 14.14
 $14.14
?#####,.#, 13131.2
13,131.2
?###, 1515
%1515
```

Programa 2: Entrada de datos.

```
10 INPUT A$, B$
20 PRINT USING A$; B$
30 GOTO 10

RUN
? !, ABCDE
A
? %%, ABCDE
AB
```

Programa 3: Lectura de datos en fichero (DATA).

```
10 REM Demo PRINT USING
20 FOR I=1 TO 5
30 READ N
40 PRINT USING "#######.##"; N
50 NEXT I
60 DATA 3.5, 500, 7800000, 41525.6, -.75
```

```
RUN
      3.50
    500.00
7800000.00
  41525.60
     -0.75
```

Cambiar la línea 40 del programa DEMO por la siguiente y ejecutarlo.

```
40 PRINT USING "**#######.##"; N

RUN
*******3.50
*****500.00
*7800000.00
***41525.60
******-0.75
```

Cambiar la línea 40 del programa DEMO por la siguiente y ejecutarlo.

```
40 PRINT USING "$$#######.##"; N

RUN
      $3.50
    $500.00
$7800000.00
  $41525.60
     -$0.75
```

Cambiar la línea 40 del programa DEMO por la siguiente y ejecutarlo.

```
40 PRINT USING "#,###,###.##"; N

RUN
        3.50
      500.00
7,800,000.00
   41,525.60
       -0.75
```

Cambiar la línea 40 del programa DEMO por la siguiente y ejecutarlo.

```
40 PRINT USING "+#######.##"; N

RUN
      +3.50
    +500.00
+7800000.00
  +41525.60
      -0.75
```

Programa 4: Formatos diversos.

```
10 A=123.456
20 PRINT USING "+###.###"; A
30 PRINT USING "+####.###"; A
40 PRINT USING "+#.###^^^^"; A
```

```
50 F$="####.###+"
60 PRINT USING F$; A

RUN
+123.456
+123.456
+1.235E+02
 123.456+
```

7.4. Sentencia LPRINT USING para salidas en impresora

Si se desea enviar la salida a la impresora se ha de modificar la sentencia PRINT USING por su homónima LPRINT USING.

Utilice las facilidades de edición de su computadora, caso de desear cambiar una, varias o todas sus sentencias PRINT USING. Sólo deberá cambiar la palabra PRINT por LPRINT.

7.5. Variantes de PRINT USING

Algunas versiones de BASIC, como las computadoras de HEWLETT PACKARD (HP-85, HP-87, HP-93-45, etc.), utilizan además de PRINT USING una sentencia nueva llamada IMAGE, unida a ella, de modo similar a RND/RANDOMIZE.

El formato de la cadena en vez de aparecer dentro de la sentencia PRINT USING, se representa en una sentencia IMAGE, por lo que la sentencia PRINT USING sólo incluirá el número de línea del formato original donde se encuentra IMAGE. La ventaja de este método reside en la simplificación que supone no escribir más que una vez el formato estándar.

Formato: PRINT USING *m, lista de expresiones*

m	Número de la línea IMAGE que contiene el formato.
lista de expresiones	Expresiones, variables numéricas o de cadenas.

Formato: IMAGE *imagen*

imagen Cadena de imagen o formato deseada.

Los separadores o especificadores de campo responden a un código específico, que si bien tiene relación con los ya estudiados, no son iguales.

Dada la restricción de estas sentencias, normalmente a computadoras HP, sólo mencionaremos algunos separadores y ejemplos, remitiendo al lector interesado a los Manuales de HP.

Separadores más importantes

X	Significa que se debe dejar un espacio en blanco.
NX	Se dejan N espacios en blanco.
" "	Todo el texto entre comillas se imprime o presenta exactamente como se ha escrito.
D	Posición de dígito.
ND	N posiciones de dígitos.

Z Posición de dígito. Las posiciones no utilizadas a la izquierda se rellenan con ceros.

NZ N posiciones de dígitos.

* Posición de dígito. Las posiciones no utilizadas a la izquierda se rellenan con *.

```
10 IMAGE "**", 4X "RESULTADOS" 4X, "**"
20 PRINT USING 10

RUN
**    RESULTADOS    **

10 IMAGE XAXXXX, AAA/
20 IMAGE ZZZ, 3X, ZZZ
30 PRINT USING 10; "I", "I*4"
40 FOR I=1 TO 3
45 PRINT USING 20: I, I*4
50 NEXT I
60 END

RUN
 I     I*4
001    004
002    008
003    012
```

PROGRAMAS RESUELTOS

1. *Aplicación de* READ/DATA.

```
100 DATA 2, 4, 8
120 READ A
140 PRINT A
160 READ B, C
200 PRINT B; C
220 END

RUN
 2
 4  8
```

2. *Aplicación de* READ/DATA.

```
100 DATA 4, 20, 60, 50
120 READ A
140 PRINT A
160 READ B, C
180 DATA 125
200 PRINT B; C

RUN
 4
 20  60
```

3. *Aplicación de* READ/DATA *con error por falta de datos en sentencia* DATA.

```
100 DATA 7, 4, 5, 3, 9
120 READ R
140 PRINT R, 2 * 3.141592 * R
160 GOTO 120
200 END

RUN
 7                  43.9823
 4                  25.1327
 5                  31.4159
 3                  18.8495
 9                  56.5487
OD ERROR IN 120
```

NOTA: OD Faltan datos debido a 140 GOTO 100.
En este caso no tiene importancia.
OD equivale al mensaje ya visto anteriormente «OUT OF DATA IN».

4. *Aplicación de* READ/DATA.

```
100 READ R
120 PRINT R, 2 * 3.141592 *R
140 GOTO 100
160 DATA 7, 4, 5, 3, 9
200 END

RUN
 7              43.9823
 4              25.1327
 5              31.4159
 3              18.8495
 9              56.5487
```

5. *Aplicación de* READ/DATA *con error por falta de datos en sentencia* DATA.

```
100 READ A, R
120 PRINT A, R, A*R*(A*R)
140 GOTO 100
160 DATA 39.2, 4, 5, 7.25, 8
180 DATA 5, 7.25, 10, 7.25
200 END

RUN
 39.2           4              24586.2
 5              7.25           1314.06
 8              5              1600
 7.25           10             5256.25
OD ERROR ON 100
```

NOTA: OD faltan datos debido a 140 GOTO 100.
En este caso no tiene importancia.

6. *Aplicación de* RESTORE.

```
100 DATA 20, 30, 40, 50, 70
110 READ A, B
120 PRINT A; B
150 RESTORE
160 FOR I=1 TO 5
180 READ C
200 PRINT C
220 NEXT I
240 END

RUN
 20  30
 20
 30
 40
 50
 70
```

7. *Aplicación de* TAB.

```
100 INPUT A, B
110 FOR I=10 TO 1 STEP -1
```

```
120 PRINT TAB(A*I+B); "*"
130 NEXT I
999 END

RUN
? 1, 3
             *
            *
           *
          *
         *
        *
       *
      *
     *
    *


? 2, 4
                        *
                      *
                    *
                  *
                *
              *
            *
          *
        *
      *
```

PREGUNTAS Y PROGRAMAS DE REPASO

1. ¿Cuál es la función de las sentencias READ, DATA y RESTORE?
2. ¿Cuál es la posición de una sentencia DATA en un programa?
3. ¿Cuántas sentencias DATA se pueden utilizar en un programa?
4. ¿Qué significa el mensaje de error OUT OF DATA IN 340?
5. ¿Cuántas líneas imprimirá el programa siguiente?

```
10 DATA 4, 6, 7, 15, 22
20 READ A
40 PRINT A, A*2
60 GOTO 20
100 END
```

6. ¿Cuántas líneas imprimirá el programa siguiente?

```
10 DATA 1, 2, 3, 4, 5, 6, 7, 8
20 READ A, B, C
30 PRINT A*B*C
40 GOTO 20
50 END
```

7. Añada la línea 45 siguiente el programa anterior y diga cuántas líneas se imprimen y su valor.

```
45 DATA 10
```

8. ¿Son indispensables las comillas en las sentencias DATA?
9. Qué sucede cuando se ejecuta el programa

```
100 DATA 15
120 READ A$
140 A$=A$+1
160 PRINT A$
```

10. Compare los resultados de estos dos programas:

```
100 DATA 400
120 READ B
140 PRINT "RESULTADO="; B
```

```
100 READ B$
120 PRINT "RESULTADO="; B$
140 DATA 400
```

11. Representar el formato de los siguientes programas:

```
10 FOR N=1 TO 4
20 PRINT SPC(4); N
30 NEXT N
99 END
```

```
10 FOR N=1 TO 4
20 PRINT TAB(6); N
30 NEXT N
99 END
```

12. Escribir un programa que realice la impresión de este dibujo:



13. Escribir un programa que realice la impresión de este dibujo:

```
* * * * * *
* * * * * *
* * *
* * * * * *
      * * *
* * * * * *
* * * * * *
```

PROGRAMAS PROPUESTOS

1. Realizar un programa que permita calcular las notas medias trimestrales de una clase de 2.ª enseñanza que tiene cinco asignaturas, y de tal modo que cada alumno pueda tener 3, 4 ó 5 notas en cada asignatura (las notas serán números de 1 a 10 con dos cifras decimales, p. e., 7.20, 8, 6.40, etc.). La nota media se obtendrá con tres decimales.

   Número de alumnos: 10.

   Los datos se deben presentar con el siguiente formato:

```
Matematicas             Notas parciales                 Nota media
-------------------------------------------------------------------
Luis Garcia Garcia      4       5       6                    5.000
Antonio Gil Ruiz        4.20    7.40    6.60                 6.061

...................     ....................                 .....
...................     ....................                 .....

Geografia               Notas parciales                 Nota media
-------------------------------------------------------------------
Luis Garcia Garcia      4       3       7       8            5.500
Antonio Gil Ruiz        6.20    4.50    7       6.40         6.025
```

CAPITULO 8

Listas y tablas

0. INTRODUCCION

Hasta este momento se han tratado variables numéricas y alfanuméricas sencillas, es decir, A1, A2$, MARTO, PEPINI, A4$, etc. Normalmente los datos a tratar en un programa representarán magnitudes de diferentes clases y en número grande; por ejemplo, los diferentes productos en un supermercado, que se pueden agrupar en familias tales como verduras, conservas, pastelería, etc., y dentro de cada familia existen una serie de elementos; en las verduras: coliflor, alcachofas, lechugas, etc.; en conservas: atún, sardinas, anchoas, etc. El tratamiento que se debe dar ha de ser específico, pues si no el número de variables que se deberían utilizar puede ser tal que haga prácticamente inoperable al programa, al tener que tratar con tantas variables. Para su resolución se recurre a las *matrices* («array*»): conjuntos ordenados de elementos dispuestos en forma de filas y columnas. Las matrices se conocen también por los nombres de *tablas* y listas.

El estudio de las matrices es una parte fundamental en la programación, ya que permitirá el tratamiento adecuado de bloques o ficheros de datos.

Después de terminar el estudio de este capítulo podrá:

— Definir una lista o tablas de variables.
— Dimensionar una variable.
— Utilizar adecuadamente las sentencias: DIM, OPTION BASE.
— Utilizar la sentencia SWAP que permite el intercambio de valores entre variables.
— Efectuar el tratamiento adecuado en pantalla o impresora de matrices.
— Liberar espacio de memoria de matrices ya utilizadas y que no se van a volver a emplear en el programa. Sentencia ERASE.
— Conocer las posibilidades de tratamiento directo de matrices en ciertos BASIC.

1. LISTAS Y TABLAS: MATRICES (ARRAYS)

Como ya se ha mencionado, hasta este punto los elementos básicos del lenguaje han sido variables simples que no contienen más que único valor en un instante dado y que tendrán las limitaciones inherentes a la definición de las variables: asociar cada elemento con una variable diferente. Sin embargo, con mucha frecuencia, es necesario poder tratar conjuntos de datos con una estructura concreta: una serie de nombres, un conjunto de números, etc., que tienen un nexo común. Haciendo un símil matemático, se necesitará representar variables tales como a_1, a_2, a_3 ..., a_i, etc., en el caso de un subíndice único, o bien a_{11}, a_{12}, a_{21}, ..., a_{ij}, etc., si se utilizan dos subíndices. Para esta representación se utilizan las matrices.

Una *lista* o *tabla* («matriz» o «array» en inglés) es un conjunto de elementos dispuestos en forma adecuada y que guardan una relación concreta entre sí. La representación de las

* En Latinoamérica se suele utilizar el término *arreglo* como traducción de *array*.

matrices se realiza mediante *variables subscritas* o *de subíndice* y pueden tener una o varias *dimensiones.* Si tiene una dimensión se denomina *lista* y si tiene dos o más dimensiones *tabla.* Las listas/tablas pueden contener variables numéricas (enteras o reales) y alfanuméricas o de cadena, pero no una mezcla de ambas.

Para la representación de las listas/tablas se recurre a las *variables con subíndice o subscritas.* Así, a_1 se representa por A(1), en donde A es el nombre de la variable y 1 es el subíndice o identificador numérico; a_i representado por A(I) es el elemento específico I de una lista de valores numéricos: A(1), A(2), A(3), ..., A(I), A(N), donde N es el índice asociado al último elemento de la lista.

Una tabla de dos dimensiones se representa con una variable con dos subíndices A(I, J), donde I representa la fila y J la columna del elemento específico. Así, elementos diferentes serían A(1,1), A(2,1), A(3,1), etc.

Ejemplo de lista:

```
A(0) A(1) A(2) ... A(N)
```

Ejemplo de tabla de dos dimensiones (M × N):

```
A(0,0)  A(0,1)  A(0,2) ... A(0,N)
A(1,0)  A(1,1)  A(1,2) ... A(1,N)
.
.
.
A(M,0)  A(M,1)  A(M,2) ... A(M,N)
```

Veamos un ejemplo concreto de aplicación, representando los productos de una tienda de comestibles, con unas listas A$, B$ y C$.

```
A$ (0)="JAMON"        B$ (0)="GALLETAS"     C$ (0)="ANCHOAS"
A$ (1)="SALCHICHON"   B$ (1)="MADALENAS"    C$ (1)="ATUN"
A$ (2)="CHORIZO"      B$ (2)="BIZCOCHOS"    C$ (2)="SARDINAS"
A$ (3)="MORTADELA"    B$ (3)="PASTELES"     C$ (3)="CABALLA"
A$ (4)="SALAMI"       B$ (4)="ROSCOS"       C$ (4)="CALAMARES"
```

En este caso se tienen tres listas distintas A$(I), B$(I), C$(I), donde I varía de 0 a 4 y se dice que tienen cinco elementos.

Se podrán representar las tres listas conjuntamente en una tabla de dos dimensiones del modo siguiente: H$(I,J), donde I varía de 0 a 4 y J de 0 a 2.

```
H$ (0,0)="JAMON"      H$ (0,1)="GALLETAS"    H$ (0,2)="ANCHOAS"
H$ (1,0)="SALCHICHON" H$ (1,1)="MAGDALENAS"  H$ (1,2)="ATUN"
H$ (2,0)="CHORIZO"    H$ (2,1)="BIZCOCHOS"   H$ (2,2)="SARDINAS"
H$ (3,0)="MORTADELA"  H$ (3,1)="PASTELES"    H$ (3,2)="CABALLA"
H$ (4,0)="SALAMI"     H$ (4,1)="ROSCOS"      H$ (4,2)="CALAMARES"
```

NOTAS:

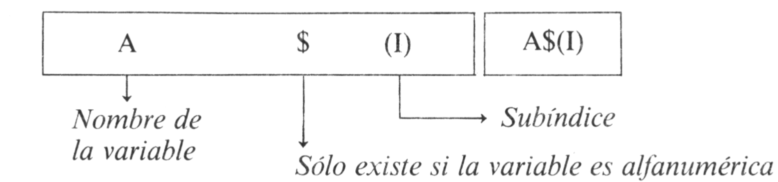
1. El concepto de lista-tabla permite representar matrices, pero no se puede confundir con el concepto de matriz en matemáticas, ya que en programación los elementos de las matrices pueden ser alfanuméricos. Por esta razón hablaremos siempre de listas y tablas dejando el nombre de matriz para su aplicación matemática.
2. Se pueden definir listas y tablas con elementos alfanuméricos sin más que añadir el símbolo $.
3. L$(I) es una variable de índice o subíndice que representa el elemento I de la lista L$.

L(I) es una variable de índice que representa el elemento I de la lista L.
L(I,J) es una variable de índice que representa el elemento específico (I, J) de la lista L.
L$(I, J) es una variable de índice que representa el elemento específico (I, J) de la lista L.

1.1. Listas

Una lista es una matriz unidimensional que se representa mediante una variable numérica o de cadena suscrita o de subíndice.

Formato:



El nombre de la variable es un nombre válido BASIC.
I es el subíndice que normalmente comienza su numeración en 0 (posteriormente se verá que existen versiones BASIC que comienzan en 1).

NOTAS:

— A$ y A$ (I) son dos variables diferentes, así como A y A (I), A$(0), A$(1), A$(2), etc., son elementos de una lista.

— A$(4) es una variable de subíndice; A3$ *no es* una variable de subíndice.
— H8 es una variable simple; H(8) es elemento octavo de una lista.
— Una lista de variables numéricas se puede representar así:

A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)
3.56	7.21	8	4.32	5.25	6	7.25	45.06

El tamaño del espacio asignado a cada variable numérica es el mismo.

— Una lista de variables de cadenas se puede representar así:

A$ (1)	VIVA ZAPATA
A$ (2)	YO QUE TU NO LO HARIA
A$ (3)	ATREVETE Y LO PASARAS MAL

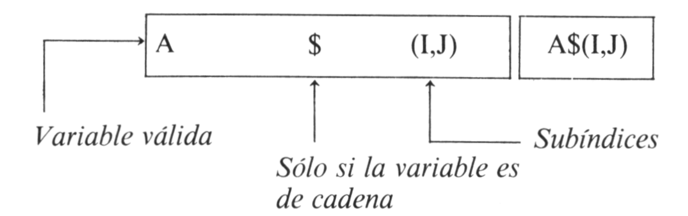
— El número de elementos de una lista es igual al número del índice superior más 1.
— Las listas, al igual que las tablas, pueden contener cadenas de caracteres, enteros o números reales, pero no una mezcla de los tres tipos.
— La longitud de las cadenas suele ser hasta 255 caracteres siempre que haya espacio disponible en memoria.
— El subíndice de las variables puede ser a su vez otra variable o expresión numérica.

1.2. Tablas

Una tabla es una matriz de varias dimensiones, aunque la más utilizadas son de dos o tres dimensiones.

Una tabla de dos dimensiones se representa mediante una variable válida y dos subíndices, uno que representa las filas y otro las columnas de dicha tabla.

Formato:



I es el número de filas de la tabla
J es el número de columnas de la tabla

La tabla A(2,5), de dimensiones 2 × 5, se representa de la siguiente forma:

A(0,0)	A(0,1)	A(0,2)	A(0,3)	A(0,4)	A(0,5)
A(1,0)	A(1,1)	A(1,2)	A(1,3)	A(1,4)	A(1,5)
A(2,0)	A(2,1)	A(2,2)	A(2,3)	A(2,4)	A(2,5)

Número de elementos 3 × 6=18.
Se podrían asignar diferentes valores a cada uno de los elementos:

```
A (0,0)=3.45     A (1,2)=7          ......
A (1,0)=7.26     A (2,3)=6          ......
A (2,0)=8.345    A (2,5)=84.32      ......
```

El tamaño (número de datos) de la tabla es el producto de la dimensión de la fila más alta incrementada en 1, multiplicada por la dimensión de la columna más alta también incrementada en 1.

A(2,3)   (2+1) × (3+1) =12 elementos

Las tablas de más de dos dimensiones se representan por A(I, J, ...).

ESTRUCTURAS DE LAS TABLAS

— La tabla unidimensional o *lista* se puede considerar como una fila o columna de elementos. La lista siguiente contiene tres elementos. Cada X representa un elemento: AS(I) o A$(I). Una lista ($n$) contiene n+1 elementos.

X	X	X

Tabla de 3 × 1=3 elementos

— La tabla bidimensional ($m \times n$) contiene (m+1) × (n+1) elementos. Cada x representa un elemento: A(I; J) o A$(I, J).

X	X	X	X
X	X	X	X

Tabla de 4 × 2=8 elementos

— La estructura de una tabla de tres dimensiones se puede imaginar como una serie de tablas de dos dimensiones.

De acuerdo con los criterios anteriores debe estructurar las tablas según sus necesidades.

2. DIMENSIONES DE LISTAS Y TABLAS: SENTENCIA DIM

Para poder utilizar una matriz, bien sea una lista o una tabla, necesitamos conocer su tamaño, esto es, el número de elementos que contiene. El tamaño máximo, como ya se ha dicho, se logra especificando el número de dimensiones y el tamaño de cada dimensión. A esta operación se la conoce como *dimensionado de la lista* o *tabla.* Cuando se dimensiona una tabla se define su tamaño máximo o físico, es decir, la cantidad total de elementos que se van a utilizar.

Si no se indica lo contrario, mediante una sentencia DIM, se supone que el intérprete de BASIC reserva espacio de memoria para 11 elementos (0, 1, 2, 3, ..., 10) en una lista; para 11 × 11=121 elementos en una tabla de dos dimensiones, etc.

NOTA:

En la mayoría de los BASIC, la sentencia DIM no es obligatoria más que cuando el número de una lista o tabla es superior a 11. Por ello, cuando trabaje con variables suscritas, si éstas no van a tener más de 11 elementos no necesita dimensionar. El valor máximo del subíndice será 10, A(10). BASIC reserva automáticamente memoria para 11 elementos.

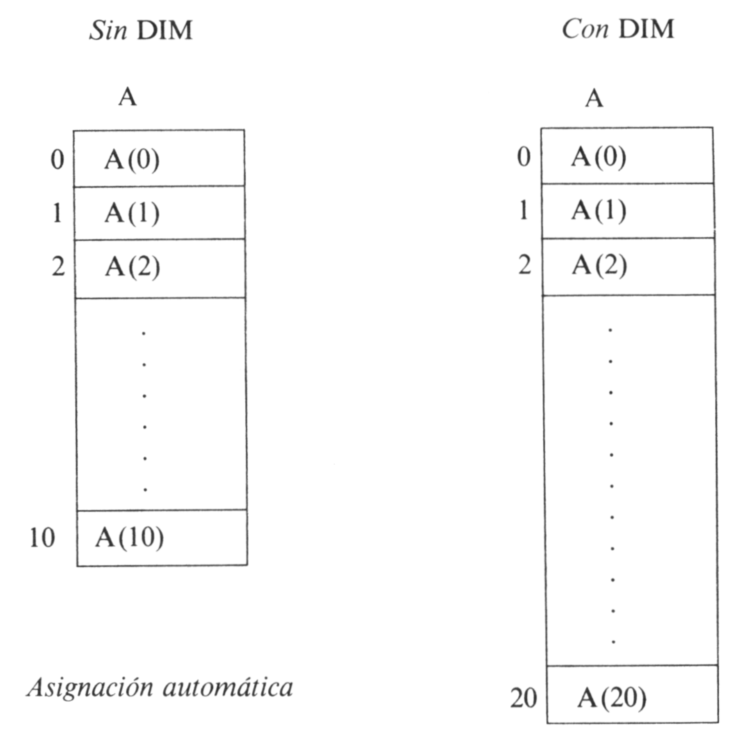
Es una buena práctica de programación dimensionar *siempre,* con lo que ahorra memoria y evita errores imprevistos por no dimensionar si son más de 11 elementos.

2.1. Sentencia DIM

La sentencia DIM se utiliza para establecer el número máximo de elementos permitidos en una lista/tabla de elementos numéricos o de cadena.

DIM A(15) reserva 16 espacios de memoria.

Así una lista A podría tener los siguientes tamaños:



Asignación automática

Asignación con DIM A(20)

La dimensión de una lista/tabla se establece situando la variable de la misma después de la sentencia DIM seguida por el tamaño de dicha lista/tabla. Por ejemplo, DIM A (20) permite que la variable A que representa la lista/tabla utilice 21 elementos, desde A(0), A(1) hasta

A(20). Algunas computadoras arrancan con elemento A(1), mientras que unas pocas HEWLETT PACKARD, IBM/PC, MBASIC, etc., pueden definir el elemento de la matriz más abajo, bien como 0, bien como 1. (Véase sentencia OPTION BASE.)

Cuando la sentencia DIM se ejecuta, la computadora activa cada uno de los valores almacenados en cada elemento de la matriz designada a cero. Sin una sentencia DIM el valor máximo del índice es normalmente 10.

Formato:

DIM *nombre (dimensión 1, dimensión 2, ...)*

nombre	Nombre de la variable numérica o alfanumérica legal.
dimensión	Valor de cada dimensión de la lista/tabla.

Así:

DIM A(25)	Reserva 26 posiciones de memoria para la lista A.
DIM C(100), Z$(20)	Reserva 101 posiciones para la variable numérica C y 21 posiciones para la lista de cadenas C$.
DIM A$(4,50)	Reserva (255) posiciones de memoria para la tabla A$.

2.2. Sentencia OPTION BASE

La mayoría de las versiones BASIC comienzan la numeración de las variables de subíndice con el subíndice 0, como ya se ha indicado. Sin embargo, hay algunas versiones de BASIC que permiten modificar el valor de la variable inicial mediante la sentencia OPTION BASE colocada antes de las sentencias DIM.

La sentencia OPTION BASE instruye a la computadora para que comience el elemento base de numeración en 0 o en 1.

OPTION BASE 0	Los subíndices comienzan en 0: A(0), A(1), A(2), ...
OPTION BASE 1	Los subíndices comienzan en 1: A(1), A(2), ...

La ventaja más importante de utilizar la sentencia OPTION BASE 1 es la localización directa de la posición a ubicación de un elemento determinado de la lista o tabla. Así, en una lista, el primer elemento es S(1), el segundo S(2), etc.

```
100 OPTION BASE 1
120 DIM A (15)
```

Al ejecutarse las sentencias 100 y 120 se reservan 15 posiciones de memoria, para A(1), A(2), ..., A(15).

```
100 OPTION BASE 0
120 DIM A (15)
```

En este caso se reservan 16 posiciones A(o), A(1), ..., A(15).

NOTAS:

— En aquellas computadoras que poseen la sentencia OPTION BASE, cuando ésta no se indica se asigna por defecto OPTION BASE 0, o sea, comienza la numeración de la variable en 0. *Se seguirá este criterio en el libro.*

— *Recuerde* que habrá de declarar la sentencia OPTION BASE antes de que aparezcan las sentencias DIM.

```
10 OPTION BASE 1
20 DIM A (10), A$ (15), B$ (25)
```

La lista A contiene 10 elementos, A$-15 y B$-25

```
10 DIM A (100)
20 DIM B (4,12), C (99,99)
```

La lista A se dimensiona a 101 elementos, la B a $5 \times 13 = 65$ elementos y la C a $100 \times 100 = 10.000$ elementos.

— Las variables de cadena se dimensionan a 255 caracteres máximo. Sin embargo, existen versiones de BASIC, como HEWLETT-PACKARD, que admiten un dimensionado implícito del número de caracteres de las variables alfanuméricas de acuerdo al siguiente formato:

```
DIM A$ (25) [40]
```

La lista A$ dimensiona 25 (26) (según el dimensionado previo con OPTION BASE) elementos alfanuméricos de 40 caracteres de longitud como máximo.

La versión BASIC INTEGER de APPLE II admite este dimensionado: DIM A$(20) cadena de 20 caracteres máximo, pero no admite tablas de cadenas.

2.3. Sentencia SWAP

La sentencia SWAP permite intercambiar el valor de una variable por otra.

Formato:

n SWAP *var*$_1$, *var*$_2$

n Número de línea.
var$_1$ Nombre de una variable (numérica o de cadena).
var$_2$ Nombre de la otra variable.

Ejemplo:

```
10 X=10 : Y=100

20 SWAP X, Y
30 PRINT X, Y

RUN
 100              10
```

Esta sentencia es muy útil para el ordenamiento o clasificación de listas y tablas en orden ascendente o descendente.

Las computadoras que no tienen la sentencia SWAP podrán reemplazar SWAP (A, B) por las siguientes sentencias:

```
100 T=A
110 A=B
120 B=T
```

Las variables a intercambiar tienen que ser del mismo tipo, numéricas o de cadenas, en caso contraio se producirá un error.

Programa de aplicación de SWAP

Los programas de clasificación que vimos en el Capítulo 8 se pueden conseguir de modo más fácil mediante esta sentencia.

```
10 INPUT "Cuantos numeros"; N
20 DIM A (N)
30 FOR I=1 TO N
40 INPUT "Numero"; A (I)
50 NEXT I
100 P=0
110 FOR I=1 TO N-1
120 IF A(I+1)<A(I) THEN SWAP A(I+1), A(I) : P=1
130 NEXT I
140 IF P=1 THEN 100
150 PRINT "Lista clasificada"
160 FOR I=1 TO N
180 PRINT A (I), I
200 NEXT I

RUN
Cuantos numeros? 6
Numero? 15
Numero? 7
Numero? 8
Numero? 4
Numero? 5
Numero? 21
Lista clasificada
 4             1
 5             2
 7             3
 8             4
 15            5
 21            6
```

3. TRATAMIENTO DE LISTAS

Vamos a ver cómo se puede obtener un buen tratamiento de listas de variables numéricas o alfanuméricas.

REPRESENTACION GRAFICA

Una lista de seis elementos dimensionada mediante DIM A(5) contiene A(0), A(1), A(2), A(3) y A(4). Su representación podría ser cualquiera de los dos esquemas siguientes:

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)

A()	A(1)	A(2)	A(3)	A(4)	A(5)

INTRODUCCION/SALIDA DE DATOS

Los valores se pueden introducir mediante INPUT, leer por READ, o bien asignar mediante LEFT.

A(1)=30 A(2)=40 A(3)=60

La introducción de valores mediante INPUT/READ permite la asignación automática a cada variable de subíndice.

```
100 FOR I=1 TO 5
120 READ A(I)
140 NEXT I
200 DATA 35, 61, 42, 101, 25
220 END
```

La ejecución del programa anterior asigna los siguientes valores:

A(1)=35 A(2)=61 A(3)=42 A(4)=101 A(5)=25

```
100 FOR I=1 TO 5
120 INPUT A(I)
140 NEXT I

RUN
?4
?7
?8
?19
?25
```

La asignación de valores es: A(1)=4, A(2)=7, A(3)=8, A(4)=19, A(5)=25.

La salida de datos en pantalla/impresora se realizaría de modo similar.

```
50 REM Entrada de datos
100 FOR I=1 TO 5
120 READ A(I)
140 NEXT I
160 DATA 35, 61, 42, 101, 25
180 REM Salida de datos
200 FOR I=1 TO 5
220 PRINT A(I);
240 NEXT I
300 END

RUN
 35  61  42  101  25
```

DIMENSION VARIABLE

En numerosas ocasiones no se conoce a priori el tamaño de una lista y entonces es preciso dimensionar *dinámicamente* la variable con otra variable o expresión numérica.

```
DIM A$ (N)
DIM B (2*N+5)
DIM A (4*N, 3/N)
```

Así:

```
100 INPUT "NUMERO DE ALUMNOS"; N
120 DIM A(N)
140 FOR I=1 TO N
160 INPUT "NOTAS"; A(I)
180 NEXT I
```

El dimensionado de la variable de la lista A se realiza en el momento de su ejecución, y se le asignan a las variables X(1), X(2), X(3), ..., X(N) los valores introducidos por teclado.

Observe que los valores de las variables comienzan en el subíndice 1 o en cualquier otro, con independencia del dimensionado.

UTILIZACION DE LOS VALORES DE UNA TABLA

Supongamos la lista A, que contiene A(1), A(2), A(3), A(4), A(5).

Con estas variables se pueden realizar todas las operaciones que ya conocemos para las variables simples:

```
K=A(1)+A(2)+A(3)
K=A(1)+A(3)
K=(A(1)+A(2)+A(3))/5
```

3.1. Normas prácticas para manejo de listas

COLOCACION DE DIM

Para que una variable quede dimensionada es preciso que la sentencia DIM correspondiente se haya ejecutado antes de la utilización de la lista.

```
10 DIM H(20)
20 FOR I=1 TO 20
30 READ H(I)
40 NEXT I
50 DATA 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
60 DATA 1, 2, 3, 14, 15, 16, 17, 18, 19, 20
```

Se ejecutaría correctamente y se asignarían a las variables H(1), H(2), H(3), etc., los valores 4, 5, 6, 7, etc.

```
10 FOR I=1 TO 20
20 READ H(I)
30 NEXT I
40 DIM H(20)
50 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
60 DATA 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
```

Aparecería el mensaje SUBSCRIT OUT OF RANGE IN 20, que significa subíndices fuera de los límites. Sólo tomaría los valores H(1), H(2), ..., a H(10), ya que la sentencia DIM se dimensionó después de la lectura de datos. Por ello le recomendamos *situar al principio del programa todas las sentencias* DIM para evitar este error.

Es aconsejable para evitar el error anterior dimensionar sistemáticamente mediante DIM todas las variables utilizadas en las diferentes listas, incluso en los casos en que su dimensión sea inferior a 11 (en este caso se ahorra además memoria).

Veamos ahora un caso en que se produce un dimensionado dinámico y éste no es correcto.

```
10 DIM K(N)
20 READ N
30 FOR I=1 TO N
40 INPUT K(I)
50 NEXT I
100 DATA 5

RUN
?10
SUBSCRIPT OUT OF RANGE IN 40
```

¿Qué ha sucedido? La varible K se dimensiona en la línea 10, pero como N no se ha inicializado, tomará el valor 0, lo que significa que se reserva una lista de dimensión 0, por lo que sólo contendrá un elemento y se obtendrá el mensaje de error.

¿Cómo corregir este error? Cambiando la línea 20 por la 10.

```
10 READ N
20 DIM K(N)
```

SUMA DE LOS ELEMENTOS DE UNA LISTA

Si se desea realizar la suma de los elementos de una lista necesitaremos un totalizador que realice las sumas parciales.

```
10 REM Lectura de datos
100 FOR I=1 TO 5
120 READ A(I)
140 NEXT I
160 REM Suma de la lista
180 S=0
200 FOR I=1 TO 5
220 S=S+A(I)
240 NEXT I                     Totalizador S = S + A(I)
260 PRINT "Suma = "; S
400 DATA 3, 4, 15, 1, 8

RUN
Suma =  31
```

Advertencia

— Desde que aparece una sentencia DIM, la variable correspondiente queda dimensionada al valor de su subíndice, aunque éste sea menor que 10.

DIM A(4) *reserva cinco posiciones de memoria*

— Si el valor de la variable que dimensiona a otra variable no es entero, se redondea dicho valor al entero más próximo.

```
100 INPUT N
120 DIM A(N)
... ...
```

```
RUN
?3.4
```
Se dimensiona
```
?5.7
```
Se dimensiona

No está permitido el redimensionado de una variable: no se puede dimensionar una variable más de una vez en un mismo programa.

```
10 DIM A(100,20), B(60)
20 DIM B(80), C(500)
```

Al ejecutarse estas sentencias se producirá un error:

DUPLICATE DEFINITION IN 20 (*Definición doble en* 20)

la razón es que se ha vuelto a dimensionar la variable B.

4. TRATAMIENTO DE TABLAS

En el párrafo 3 hemos visto el tratamiento de listas. Se va a considerar ahora cómo trabajar con tablas (matrices de más de una dimensión).

Supongamos que se ha dimensionado la tabla A mediante DIM A(3,4). Esta sentencia nos reserva $4 \times 5=20$ posiciones de memoria, de modo que se pueden representar por el siguiente esquema:

A(0,0)	A(0,1)	A(0,2)	A(0,3)	A(0,4)
A(1,0)	A(1,1)	A(1,2)	A(1,3)	A(1,4)
A(2,0)	A(2,1)	A(2,2)	A(2,3)	A(2,4)
A(3,0)	A(3,1)	A(3,2)	A(3,3)	A(3,4)

Esta tabla tiene 4 filas y 5 columnas y se dice que tienen 2 dimensiones o 2 índices; el primero puede tomar 4 valores (0, 1, 2, y 3) y el segundo 5 valores (0, 1, 2, 3, 4).

INTRODUCCION/SALIDA DE DATOS

Al igual que con las listas, la introducción de valores se realiza mediante las sentencias LET, INPUT y READ.

- Mediante LET:

```
100 LET A(1,1)=5.4
120 LET A(1,3)=4
140 LET A(1,2)=6
```

- Mediante READ:

```
100 READ A(1,1)
110 READ A(1,2)
120 READ A(2,1)
140 READ A(2,2)
200 DATA 4, 6, 8, 14
```

Al ejecutarse estas líneas, los valores que tomarían las variables subscritas serían:

```
A(1,1)=4          A(1,2)=6
A(2,1)=8          A(2,2)=14
```

- Mediante sentencias INPUT:

```
100 INPUT A(1,1)
120 INPUT A(1,2)
140 INPUT A(2,1)
160 INPUT A(2,2)
```

Sin embargo, las formas anteriores no son las usuales para la introducción de datos y éstos se introducen *normalmente* mediante bucles FOR-NEXT anidados en número igual al de dimensiones de la tabla.

Ejemplo 1: Lectura de datos de un fichero DATA.

```
100 FOR I=1 TO 3
120 FOR J=1 TO 5
140 READ K(I,J)
160 NEXT J
180 NEXT I
200 DATA 14, 3, 2, 12, 6, 4, 8
220 DATA 1, 0, 8, 4, 3, 2, 13, 1
```

Al ejecutar este programa se han introducido los valores en la tabla:

← J →

I ↕					
	14	3	2	12	6
	4	8	1	0	8
	4	3	2	13	1

Si se intercambian los índices de los bucles, el programa y tabla leída serían:

```
100 FOR I=1 TO 5
120 FOR J=1 TO 3
140 READ K(I,J)                        14    3    2
160 NEXT J                             12    6    4
180 NEXT I                             8     1    0
200 DATA 14, 3, 2, 12, 6, 4, 8         8     4    3
220 DATA 1, 0, 8, 4, 3, 2, 13, 1       2     13   1
```

Ejemplo 2: Salida de datos.

El primer programa del ejemplo anterior se puede ampliar para imprimir la tabla leída por los bucles I, J del modo siguiente:

```
50 REM Lectura de datos
100 FOR I=1 TO 3
120 FOR J=1 TO 5
140 READ K(I,J)
180 NEXT J
```

```
200 NEXT I
220 DATA 14, 3, 2, 12, 6, 4, 8
240 DATA 1, 0, 8, 4, 3, 2, 13, 1
260 REM Salida de datos
280 FOR I=1 TO 3
300 FOR J=1 TO 5
320 PRINT K(I,J);
340 NEXT J
380 PRINT
400 NEXT I

RUN
 14  3  2  12  6
  4  8  1   0  8
  4  3  2  13  1
```

Obsérvese que se ha introducido la línea 380 PRINT. La razón es que si no se hace se escribiría toda la tabla en una sola línea. *Hágalo y trate de deducir el motivo.* Sustituya la línea 320 por 320 PRINT K (I, J) y borre la línea 380. ¿Qué se obtiene?

Recuerde: Es conveniente dimensionar todas las listas o tablas a su valor adecuado. En caso de no conocer a priori las dimensiones exactas será conveniente realizar el dimensionado de modo aproximado, *por supuesto* siempre por exceso, para evitar errores. Ahorrará espacio de memoria con el dimensionado, incluso para índices menores de 10.

SUMA DE LOS ELEMENTOS DE UNA TABLA

Se pueden presentar tres casos:

a) *Suma de todos los elementos*

Modifique el programa anterior para obtener la suma de todos sus elementos del modo siguiente y ejecútelo.

```
270 S=0
280 FOR I=1 TO 3
300 FOR J=1 TO 5
320 S=S+K(I,J)
340 NEXT J
360 NEXT I
400 PRINT "SUMA ="; S

RUN
SUMA = 81
```

b) *Suma de los elementos de una fila*

```
280 FOR I=1 TO 3
300 FOR J=1 TO 5
320 S(I)=S(I)+K(I,J)
340 NEXT J
360 PRINT "SUMA FILA"; I; "="; S(I)
380 NEXT I
400 END

RUN
SUMA FILA 1 = 37
SUMA FILA 2 = 21
SUMA FILA 3 = 23
```

c) *Suma de los elementos de una columna*

Se cambia el orden de los bucles.

```
280 FOR J=1 TO 5
300 FOR I=1 TO 3
320 S(J)=S(J)+K(I,J)
340 NEXT I
360 PRINT"SUMA COLUMNA"; J; "="; S(J)
380 NEXT J
400 END

RUN
SUMA COLUMNA 1 = 22
SUMA COLUMNA 2 = 14
SUMA COLUMNA 3 = 5
SUMA COLUMNA 4 = 25
SUMA COLUMNA 5 = 15
```

5. EJERCICIOS PRACTICOS DE PROGRAMACION

Dada la importancia que el tratamiento de listas y tablas tiene en cualquier programa, se recogen en este párrafo ejercicios de dificultad creciente y variados a fin de que el lector pueda finalizar con esta técnica.

Ejercicio 1: Lectura e impresión de una lista en diferentes líneas o en una sola línea.

a)

```
100 DIM A(7)
120 FOR K=1 TO 7
140 READ A(K)
160 NEXT K
180 FOR K=1 TO 7
200 PRINT A(K)
220 NEXT K
300 DATA 7, 14, 3, 5, 4, 8, 9

RUN
 7
 14
 3
 5
 4
 8
 9
```

Cambie la línea 200 por 200 PRINT A(K); y ejecute el programa:

```
RUN
 7  14  3  5  4  8  9
```

Cambie la línea 200 por 200 PRINT A(K); TAB (5*K+5); y ejecute el programa:

```
RUN
 7          14          3          5          4          8          9
```

Ejercicio 2: Copia de una lista A en otra lista B.

```
10 DIM A(6), B(6)
20 FOR I=1 TO 6
30 READ A(I)
40 NEXT I
50 FOR J=1 TO 6
60 B(J)=A(J)
70 PRINT A(J), B(J)
80 NEXT J
100 DATA 4, 7, 8, 3, 2, 1

RUN
 4              4
 7              7
 8              8
 3              3
 2              2
 1              1
```

Cambie la línea 60 por B(J)=A(7-J) y ejecute el programa.

```
RUN
 4              1
 7              2
 8              3
 3              8
 2              7
 1              4
```

Ejercicio 3: Obtención del número más grande almacenado en sentencias DATA.

```
100 DIM A(10)
110 FOR N=1 TO 10
120 READ A(N)
130 NEXT N
140 B=A(1)
150 FOR I=2 TO 10
160 IF A(I)>B THEN 180
170 GOTO 190
180 B=A(I)
190 NEXT I
200 PRINT "NUMERO MAYOR": B
210 DATA 31, 8, 5, 6, 42, 7, 15, 4, 9, 10

RUN
NUMERO MAYOR 42
```

Las líneas 160, 170, 180 pueden ser sustituidas por 160 IF A(I)>B THEN B=A(I)

Se leen todos los valores y se cargan en la tabla A, luego se asigna A(1) al valor B. A continuación se examinan los restantes valores de la tabla A para encontrar un valor superior a B, en este caso el valor encontrado se asigna a B. El último valor de B es el valor buscado y se imprime.

El programa anterior se puede modificar para ampliar cuando se desee el total de números hasta un cierto límite que fijaremos en 200. Como ejemplo añadimos otra línea DATA con nuevos números y una marca fin de fichero 0.

```
220 DATA 7, 14, 31, 37, 121, 224, 0
```

El programa quedaría así:

```
100 DIM A(200)
110 FOR N=1 TO 200
120 READ X
125 IF X=0 THEN 150
127 A(N)=X
130 NEXT N
140 B=A(1)
150 FOR I=2 TO N-1
160 IF A(I)>B THEN 190
170 GOTO 190
180 B=A(I)
190 NEXT I
200 PRINT "NUMERO MAYOR"; B
210 DATA 31, 8, 5, 6, 42, 7, 15, 4, 9, 10
220 DATA 7, 14, 31, 37, 121, 224, 000
230 END
```

Ejercicio 4: Obtención de tablas matemáticas.

Tabla de multiplicación

```
10 FOR I=1 TO 10
20 FOR J=1 TO 10
30 PRINT I; "POR"; J; "="; I*J
40 NEXT J
50 NEXT I

RUN
 1 POR 1 = 1
 1 POR 2 = 2
 1 POR 3 = 3
 ...........
 ...........
 2 POR 1 = 2
 2 POR 2 = 4
 ...........
 ...........
 ...........
 10 POR 10 = 100
```

Tabla de potencias de números enteros

```
100 FOR I=1 TO 10
120 PRINT I, I^2, I^3, I^4
130 NEXT I

RUN
 1            1            1            1
 2            4            8            16
 3            9            27           81
 4            16           64           256
 5            25           125          625
```

Ejercicio 5: Encontrar el número mayor de una lista.

Número más grande: MAYOR.
Números de la lista: X.

Los pasos a seguir son:

- Leer el primer número, se le asigna a la variable MAYOR.
- Leer el siguiente número, X.
- Si X es mayor que MAYOR, entonces sustituya el valor de MAYOR con el valor de X.

```
100 READ MAYOR
110 FOR N=1 TO 6
120       READ X
130       IF N>MAYOR THEN MAYOR=X
140 NEXT N
150 PRINT "El numero mayor es"; MAYOR
160 DATA 24,57,15,78,95,45
```

6. SUBRUTINAS DE CLASIFICACION (ORDENACION) NUMERICA O ALFABETICA DE LISTAS

Por la gran importancia que tienen en la programación la ordenación numérica creciente o decreciente de números o palabras, dedicamos todo este párrafo a exponer diversos métodos de clasificación u ordenación («sort» en inglés), que si no son todos los existentes, sí son lo suficientemente representativos para que puedan ser utilizados por el lector con éxito.

Los números, palabras o cadenas a ordenar o clasificar por orden creciente o decreciente se introducen en una lista A(1), A(2), A(3)... A(N). La organización de una lista de datos ordenados al azar, de modo que queden en el orden deseado, será realizado mediante algoritmos de clasificación.

Por ejemplo, si tiene la lista 3, 7, 2, 4, 14, 21, 40, 13, la clasificación en orden creciente sería 2, 3, 4, 7, 13, 14, 21, 40. La clasificación en orden decreciente sería 40, 21, 14, 13, 7, 4, 3, 2.

6.1. Clasificación en orden creciente

Este método se conoce como método de *mínimos*, que consiste en buscar el número mínimo (más pequeño de la lista) y situarlo como primer elemento de la lista después de buscar el mínimo de la lista restante, y se coloca en segundo lugar, y así sucesivamente. Si la lista tiene N elementos se realizarán N−1 operaciones, ya que cuando falte un elemento por computadora éste será el último.

El programa correspondiente es:

Subrutina 1:

```
100 REM Clasificacion creciente
110 INPUT "Cuantos numeros"; N
120 DIM A(N)
130 REM Entrada de datos
140 FOR I=1 TO N
150 INPUT A(I) : REM Lista a ordenar
160 NEXT I
170 REM Subrutina de clasificacion.
200 FOR I=1 TO N
210 M=A(I)
215 K=I
220 FOR J=I TO N
230 IF A(J)<M THEN K=J:M=A(J)
240 NEXT J
250 B=A(I):A(I)=A(K):A(K)=B
260 PRINT M
270 NEXT I
```

En las líneas 150, 230 y 250 se han escrito varias sentencias a una sola línea.
Una variante de este programa podría ser modificar las líneas 170 en adelante.

Subrutina 2:

```
200 FOR I=1 TO N-1
210 K=I
220 M=A(I)
230 FOR J=I+1 TO N
240 IF M<=A(J) THEN 270
250 K=J
260 M=A(J)
270 NEXT J
280 A(K)=A(I)
290 A(I)=M
300 NEXT I
305 REM Imprimir lista
310 FOR I=1 TO N
320 PRINT A(I)
330 NEXT I
```

6.2. Clasificación de burbujas

Este método se conoce de este modo (*buble sort*, en inglés) debido a que durante el proceso de clasificación las cantidades se van «añadiendo» dentro de la tabla hasta colocarse en su posición, de igual forma que las burbujas de las bebidas espumosas.

Este procedimiento es tal vez el más utilizado y se realiza comparando en primer lugar los dos elementos de la tabla A(1) y A(2). Si el segundo es más pequeño que el primero se intercambian, o sea, en A(2) se pone ahora el mayor de los dos. Seguidamente se comparan el segundo y el tercero, A(2) y A(3), volviendo a hacer que A(3) sea el mayor de los dos. Este proceso se continúa por toda la tabla hasta llegar a la última pareja de elementos, con lo que se tendrá el mayor elemento colocado en último lugar. A continuación se realiza una segunda vuelta, y así hasta que todas las cantidades estén en el orden correcto.

Ejemplo:	Sea la lista	18	13	16	10
	Primera comparación	13	18	16	10
	Segunda comparación	13	16	18	10
	Tercera comparación	13	16	10	18
	Segunda vuelta	13	10	16	18
	Segunda comparación	10	13	16	18

Se efectúan N−1 exámenes de la tabla que forman el bucle I y cada vez se efectúan N−1 comparaciones, que se realiza con un bucle interno J.

Consideremos la misma entrada de datos del apartado 6.1. La subrutina de ordenación sería:

Subrutina 3:

```
200 FOR I=1 TO N-1
210 FOR J=1 TO N-1
220 IF A(J)<=A(J+1) THEN 260
230 H=A(J)
240 A(J)=A(J+1)
250 A(J+1)=H
260 NEXT J
270 NEXT I
280 FOR I=1 TO N : PRINT A(I) : NEXT I
```

Esta subrutina se puede mejorar, dado que al ejecutar el bucle «I» se fija un elemento de la tabla y, por consiguiente, no necesita examinar. Se puede modificar de la forma siguiente:

Subrutina 4:

```
200 FOR I=1 TO N-1
210 P=0
220 FOR J=1 TO N-1
230 IF A(J)<=A(J+1) THEN 280
240 H=A(J)
250 A(J)=A(J+1)
260 A(J+1)=H
270 P=1
280 NEXT J
290 IF P=0 THEN 400
300 NEXT I
```

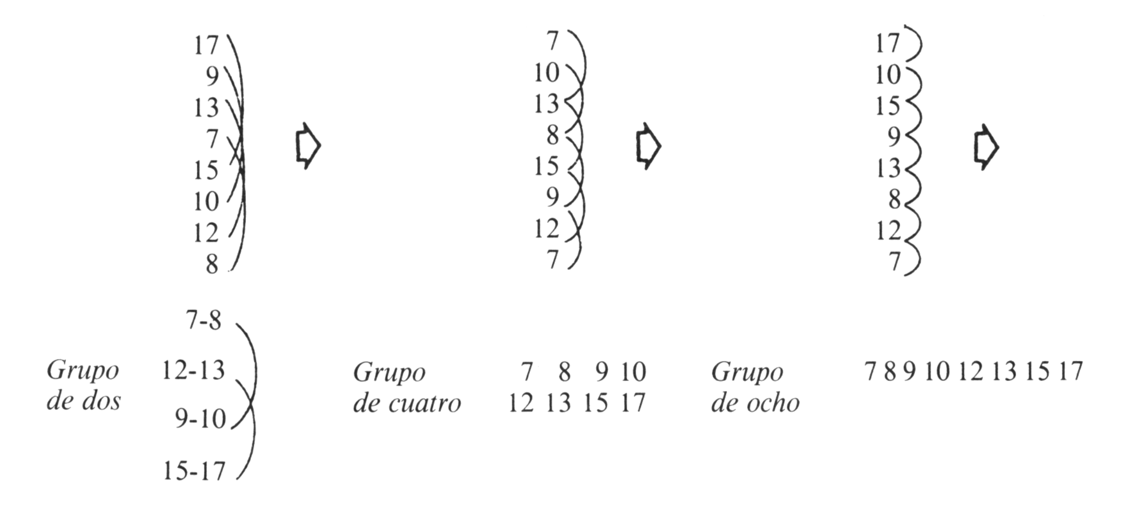
La varible P tiene por objeto averiguar si ha habido cambio durante la secuencia, ya que si se ha recorrido toda la tabla y no se ha intercambiado ningún elemento la tabla está clasificada.

6.3. Clasificación por algoritmo de Shell

El método de burbuja es fácil de realizar. Sin embargo, es *lento.* Mediante el algoritmo de «Shell» (Método «Shell-Short» o de «Concha») se obtiene el método de clasificación más rápido de los tres expuestos hasta ahora.

El método se basa en una sucesión de clasificaciones. Se comparan los elementos dos a dos de la forma siguiente: A(I) con A(I+n/2), y se les ordena. A continuación se fusionan los grupos de dos elementos en grupos de cuatro, después los grupos de cuatro en grupos de ocho, hasta que se obtiene la clasificación final.

Ejemplo: Clasificar la lista de números 17 9 13 7 15 10 12 8



Subrutina 5:

```
200 F=N
210 F=INT((F+1)/2)
220 FOR J=1 TO N-F
230 I=J
240 IF A(I)<=A(I+F) THEN 280
250 C=A(I)
260 A(I)=A(I+F)
```

```
270 A(I+F)=C
280 NEXT J
290 IF F=1 THEN 330
300 F=F-1
310 GOTO 220
320 REM Salida de datos
330 FOR I=1 TO N
340 PRINT A(I);
350 NEXT I
```

7. LIBERACION DE MEMORIA (ERASE)

La sentencia ERASE («borra») una matriz del programa de la memoria central, en el sentido de eliminar su contenido y liberar el espacio de memoria inicialmente reservado por la sentencia DIM.

Formato: ERASE *nombre matriz*1, *nombre matriz*2, ...

Después de que se ha borrado una/s matriz/ces, el espacio de memoria que ha sido asignado a la/s matriz/ces se puede utilizar para otros propósitos.

```
100 ERASE NOMBRE$(50),CALIFICACIONES(50,4)
```

esta línea borra las dos matrices NOMBRE$ y CALIFICACIONES.

Hay diferentes razones por las que puede necesitar utilizar ERASE. En primer lugar, si una matriz ya ha sido empleada y no se va a utilizar más, se puede borrar esa matriz y aprovechar el espacio de memoria liberado para otras tareas.

Una segunda razón para utilizar ERASE es redimensionar una matriz. Si trata de redimensionar una matriz sin borrarla primero, se producirá un error «Duplicate Definition» (Definición duplicada).

La orden CLEAR (Capítulo 10) se utiliza para borrar todas las variables del área de trabajo.

En el ejemplo siguiente, se observa cómo el espacio o zona de memoria ocupada por la matiz A es recuperada por BASIC, mediante la sentencia ERASE, y a continuación dimensionada nuevamente esa matriz.

```
10 DIM A(200)
20 PRINT FRE(O)              'Espacio libre de memoria.
30 ERASE A                   'Borrado de memoria de la matriz A.
40 PRINT FRE(O)              'Espacio libre de memoria.
RUN
 42377                       Espacio libre de memoria con A(200) en memoria.
 43391                       Espacio libre de memoria con A(200 borrada de memoria.
```

En el Capítulo 10 se analizará la sentencia FRE para la obtención del espacio de memoria libre.

La sentencia ERASE se encuentra en versiones de MBASIC, IBM/PC, MSX y Amstrad, entre otras computadoras.

8. SENTENCIAS ESPECIALES DE MANIPULACION DE MATRICES (MAT)

Existen computadoras específicas para cálculos científicos (Sistema HP) 9845 de Hewlett Packard, VAX de Digital, Textronix) que admiten en su BASIC sentencias especiales de manipulación de matrices que simplifican considerablemente los cálculos de listas y tablas. Aunque

no son frecuentes en las microcomputadoras, dado su interés desde el punto de vista técnico y de gestión, se considerará a continuación.

Las operaciones con matrices se simplifican considerablemente, dado que los cálculos se efectúan directamente y no elemento a elemento.

8.1. Lectura y escritura de matrices

Las sentencias que permiten la lectura/escritura de matrices son:

```
MAT READ
MAT INPUT
MAT PRINT
```

Sentencia MAT READ

Permite la lectura de todos los valores de una lista o tabla a partir de sentencias DATA situadas en el programa:

Formato: | *n MAT READ Matriz* |

n	Número de línea.
Matriz	Nombre y dimensiones de la matriz. Si éstas no son específicas se supone que ha sido dimensionada previamente.

Ejemplo 1:

```
10 DIM A(7)
20 MAT READ A
```

La sentencia DIM permite a la variable A utilizar ocho elementos de la lista A, denominados A(0), A(1), A(2), ..., A(7).

```
10 DIM A(3,4)
20 MAT READ A
```

La computadora lee una tabla de tres filas por cuatro columnas o sea doce elementos y los asigna a la variable A.

Ejemplo 2:

```
100 DIM A(3,4)
110 MAT READ A
120 DATA 1, -4, 0, -4
130 DATA 6, 4, 1, 8
140 DATA 3, 5, 6, 1
150 END
```

La matiz A se lee fila por fila y resulta:

$$A = \begin{bmatrix} 1 & -4 & 0 & -4 \\ 6 & 4 & 1 & 8 \\ 3 & 5 & 6 & 1 \end{bmatrix}$$

Si no existiera la sentencia MAT READ A, la línea 110 deberá ser sustituida por:

```
FOR I=1 TO 3
FOR J=1 TO 4
READ A(I,J)
NEXT J
NEXT I
```

Sentencia MAT INPUT

Permite la introducción de valores por teclado y su asignación a los diferentes elementos de una matriz.

```
10 DIM A(5)
20 MAT INPUT A
```

La sentencia DIM permite rellenar la lista A, desde A(0), A(1) a A(5).

Formato: | *n MAT INPUT Matriz* |

n Número de línea.
Matriz Nombre y dimensiones de la matriz. Si éstas no se especifican se supone que ha sido dimensionada previamente.

Cuando la sentencia MAT INPUT se ejecuta, la computadora imprime ?, indicando que está preparada para recibir el primer elemento de la matriz. Presentará el signo ? mientras queden elementos por introducir.

```
10 DIM A(2,3)
20 MAT INPUT A
```

La computadora asigna valores a los elementos variables de la matriz A(1,1), A(1,2), A(1,3), y después a(2,1), A(2,2) y A(2,3).

Si no *existiera* la sentencia MAT INPUT A, la línea 20 se deberá sustituir por:

```
FOR I=1 TO 2
FOR J=1 TO 3
INPUT A(I,J)
NEXT J
NEXT I
```

Sentencia MAT PRINT

La sentencia MAT PRINT permite la impresión de los valores almacenados en una matriz. El número de elementos impresos viene definido en el valor asignado a la matriz al dimensionarla mediante DIM.

```
10 REM Demo MAT
20 DIM A(5)
30 FOR H=1 TO 5
40 A(H)=H
50 NEXT H
60 MAT PRINT A;
70 PRINT "Fin"

RUN
 1  2  3  4  5
```

Cambie la línea 60 por MAT PRINT A y ejecute el programa.

```
RUN
 1
 2
 3
 4
 5
Fin
```

La sentencia MAT PRINT de la línea 60 equivale a

```
FOR I=1 TO 5
PRINT A;
NEXT I
```

8.2. Sentencias de operaciones con matrices

Suma y resta

Formato: MAT C=A+B

La operación exige que las matrices tengan iguales dimensiones.

Formato: MAT C=A−B

Multiplicación

Formato: MAT C=A * B

Multiplicación de una matriz por un escalar

Formato: MAT C= *(expresión aritmética)* * *A*

Transpuesta de una matriz

Formato: MAT B=TRN (A)

Inversión

Formato: MAT B=INV (A)

Matriz unidad

Todos los elementos de la matriz se ponen a 1.

Formato: MAT A=CON *Ejemplo:* MAT A=CON (3,3) da

$$A=\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

Matriz cero

Todos los elementos de la matriz se ponen a 0.

Formato: MAT A=ZER *Ejemplo:* MAT A=ZER (2,2) da

$$A=\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$$

Matriz identidad

Una matriz identidad contiene unos en la diagonal principal y ceros en los restantes elementos.

Formato: MAT A=IDN *Ejemplo:* MAT A=IDN (4,4)

$$A=\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

8.3. Ejemplos de aplicación de sentencias MAT

1. *Lectura e impresión de una tabla*

```
10 DIM A(3,3)
20 MAT READ A
30 MAT PRINT A;
40 DATA 4, 5, 7, 8, 9, 6
50 DATA 3, 1, 4

RUN
 4  5  7
 8  9  6
 3  1  4
```

Verifique la línea 30 por 30 MAT PRINT A. ¿Qué se obtendría?

2. *Lectura e impresión de una tabla*

```
10 DIM A(3,4)
20 MAT READ A
30 MAT PRINT A;
40 DATA 7, 8, 9, 1, 2, 3, 4, 5, 6
50 DATA 1, 0, 3, 7, 8, 9, 4, 5, 7
60 DATA 15, 5, 6, 7, 8, 1

RUN
 7  8  9  1
 2  3  4  5
 6  1  0  3
```

Cambie la línea 10 por 10 DIM A(6,4) y ejecute el programa:

```
RUN
 7  8  9  1
 2  3  4  5
 6  1  0  3
 7  8  9  4
 5  7  15  5
 6  7  8  1
```

3. *Debe hacer suficientes valores en la sentencia* DATA

```
10 DATA 3, 5, 9, 8, 3, 4, 5, 6, 4, 7, 8, 2
20 DATA 4, 9, 1, 2, 8, 1, 7, 6, 9, 3, 1, 0
30 DIM A(6,6)
40 MAT READ A
50 MAT PRINT A;
60 END

RUN
OUT OF DATA IN 40
```

4. *Se pueden imprimir varias tablas a la vez*

```
10 DATA 3, 5, 4, 8, 3, 9, 5, 1, 4, 9, 7, 3
20 DATA 0, 1, 6, 1, 4, 5, 1, 3, 3
30 DIM M(3,3), N(2,4), P(2,2)
40 MAT READ A, B, C
50 MAT PRINT A; B; C
60 END

RUN
 3  5  4
 8  3  9
 5  1  4

 9  7  3  0
 1  6  1  4

 5  1
 3  3
```

5. *Uso de* MAT INPUT:

```
100 DIM A(4,3)
110 MAT INPUT A
120 MAT PRINT A;
140 END

RUN
?5, 7, 8, 9, 6, 1, 2, 7, 3, 5, 1, 0
 5  7  8
 9  6  1
 2  7  3
 5  1  0
```

6. *Redimensionado mediante* MAT INPUT:

```
10 DIM A(5,5)
20 MAT INPUT A(2,2)
30 MAT PRINT A;
40 END

RUN
?1, 4, 7, 0
 1  4
 7  0
```

7. *Comparación de entrada/salida de datos (matrices) con sentencias* MAT *y sin ellas.*

a) Sin sentencias MAT

```
100 REM entrada / salida sin sentencias MAT
110 DIM A(2,4)
120 FOR I=1 TO 2
130 FOR J=1 TO 4
140 READ A(I,J)
150 NEXT J
160 NEXT I
170 REM impresion de datos
180 FOR I=1 TO 2
190 FOR J=1 TO 4
200 PRINT A(I,J)
210 NEXT J
220 NEXT I
230 REM datos
240 DATA 24,56,-37,15
250 DATA 8,9,221,-40
```

b) Con sentencias MAT

```
100 REM entrada / salida con sentencias MAT
110 DIM A(2,4)
120 MAT READ A
130 MAT PRINT A
140 DATA 24,56,-37,15
150 DATA 8,9,221,-40
```

PROGRAMAS RESUELTOS

1. *Obtención de la nota de una asignatura, así como de las notas más altas y más bajas de la lista de clase. Máximo número de alumnos 50:*

```
80 OPTION BASE 1
90 DIM A(50)
100 REM entrada calificación
110 INPUT "cuantos alumnos ";N
120 FOR I=1 TO N
130 PRINT "Alumno:";I
140 INPUT A(I)
150 NEXT I
160 REM Suma de notas
180 S=0
190 FOR I=1 TO N
200 S=S+A(I)
210 NEXT I
220 M = S/N
230 PRINT "Media de clase =";M
240 REM Nota mayor-menor
250 H=A(1)
260 B=A(1)
270 FOR I=2 TO N
280 IF B<A(I) THEN 300
290 B=A(I)
300 IF H>A(I) THEN 320
310 H=A(I)
320 NEXT I
330 PRINT "Nota más alta: ";H
340 PRINT "Nota más baja: ";B
999 END
```

2. *Obtención de las notas medias de un curso de BUP por alumno y por asignatura para el caso de seis asignaturas.*

   En este caso habrá que aplicar los ejemplos del párrafo 4 de suma de elementos de una tabla y habrá que sumar por filas para obtener la media del alumno y por columnas para obtener la media de las asignaturas.

   Ejemplo:

ALUMNO	*ASIG. 1*	*ASIG. 2*	*ASIG. 3*	*ASIG. 4*	*ASIG. 5*	*ASIG. 6*
1	4	6	7	8	9	10
2	5	6	4	2	4	8
3	3	7	3	1	6	9

Suma de columnas para media de asignaturas.
Suma de filas para media de alumnos.

```
10 DIM K(6,50) : REM Dimensionando 6 asig. y 50 alumnos
20 INPUT "Numero de alumnos"; N
30 FOR I=1 TO N
```

```
40 FOR J=1 TO 6
50 INPUT K(I,J)
60 NEXT J
70 NEXT I
80 REM Notas medias alumnos (FILAS)
90 FOR I=1 TO N
100 S=0
110 FOR J=1 TO 6
120 S=S+K(I,J)
130 NEXT J
140 PRINT "Alumno"; I; "="; S/6
150 NEXT I
160 REM Media asignatura (COLUMNAS)
170 FOR J=1 TO 6
180 S=0
190 FOR I=1 TO N
200 S=S+K(I,J)
210 NEXT I
220 PRINT J, S/N
230 NEXT J
300 END
```

3. *Tratamiento de matrices matemáticas.*

En las versiones de BASIC más frecuentes no existen sentencias específicas para tratamiento de matrices, por lo que si se necesita será preciso realizarlo mediante tablas.

Matriz unidad

La matriz unidad es aquella en la que todos los elementos de la diagonal principal son igual a la unidad.

$$A = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

los elementos cumplen: A(I,I)=1
A(I,J)=0

El programa que permite obtener una matriz unidad es:

```
100 DIM A(15,15)
120 FOR I=1 TO 15
130 FOR J=1 TO 15
140 A(I,J)=0
150 IF I=J THEN A(I,J)=1
160 PRINTA(I,J);
170 NEXT J
180 PRINT
190 NEXT I
200 END
```

Multiplicación de una matriz por un escalar

$$AxK = \begin{bmatrix} A(1,1) & A(1,2) & A(1,3) \dots \\ A(2,1) & A(2,2) & A(2,3) \dots \\ A(3,1) & A(3,2) & A(3,3) \dots \end{bmatrix} x\ K$$

El programa sería

```
100 FOR I=1 TO N
120 FOR J=1 TO N
130 A(I,J)=K*A(I,J)
140 NEXT J
150 NEXT I
```

Se multiplica cada elemento de la matriz por K y se asigna la matriz B.

Transposición de una matriz

```
100 FOR I=1 TO M
110 FOR J=1 TO N
120 B(I,J)=A(I,J)
130 NEXT J
140 NEXT I
```

Suma de matrices

```
100 FOR K=1 TO N1
110 FOR J=2 TO N2
120 FOR I=3 TO N3
130 C(I,J,K)=A(I,J,K)+B(I,J,K)
140 NEXT I
150 NEXT J
160 NEXT K
```

PREGUNTAS Y PROGRAMAS DE REPASO

1. ¿Qué es una tabla? ¿Cómo se representa?
2. ¿Para qué sirve una sentencia DIM?
3. ¿Qué hacen estas sentencias?

   a) `DIM A$(50), M$(25), C(101)`
   b) `DIM C1(45), X8(4), K(4,17)`

4. Se tiene una lista de 500 números. Cómo se puede calcular la suma.
5. Qué sucede al ejecutar el programa

```
100 DIM T(10)
120 INPUT N
130 FOR K=1 TO N+1
140 B(K)=K^2
150 PRINT B(K)
160 NEXT K
```

   Si N vale: 3, 5, 15

6. ¿Qué valores pueden tomar los índices si la tabla A está dimensionada por H (4,121)?
7. Escribir un programa que cambie cada elemento de la tabla por un cero. Imprimir la tabla original y la nueva.
8. ¿Qué se imprime si se ejecuta el siguiente programa?

```
100 READ N
120 FOR H=1 TO N
130 READ X(H)
140 NEXT H
150 FOR H=1 TO N
160 IF X(H)<0 THEN 180
170 PRINT X(H);
180 NEXT H
200 DATA 8
220 DATA 23, 44, -5, 7, 6, 4, 13, 27
300 END
```

9. Escriba un programa que llene una tabla 15 por 15 con ceros.
10. Hallar el valor medio de una lista de 50 números que se encuentran almacenados en un sentencia DATA.
11. ¿Cuál será la salida (valor de N) si ejecuta el siguiente programa?

```
5 DIM B(6)
10 FOR I=1 TO 6
20 READ B(I)
```

```
30 NEXT I
40 S1=0
50 S2=0
55 FOR I=1 TO 6
60 S1=S1+B(I)
70 S2=S2+B(I)^2
80 NEXT I
90 N=S2-S1
100 PRINT N
110 DATA 2, 1, 3, 1, 2, 1
```

12. ¿Cuál será la salida (valor de S) si ejecuta el siguiente programa?

```
10 DIM H(4,4)
20 FOR I=1 TO 4
30 FOR J=1 TO 4
40 READ H(I,J)
50 NEXT J
60 NEXT I
70 S=0
80 FOR I=1 TO 4
90 S=S+H(I,S-I)
100 NEXT I
110 PRINT S
120 DATA 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
130 DATA 20, 21, 22, 23
199 END
```

13. ¿A qué sentencia MAT equivale el siguiente programa?

```
100 FOR I=1 TO 4
110 FOR J=1 TO 6
120 A(I,J)=0
130 NEXT J
140 NEXT I
```

14. ¿A qué sentencia MAT equivale el siguiente programa?

```
100 FOR I=1 TO 5
110 FOR J=1 TO 10
120 C(I,J)=A(I,J)+B(I,J)
130 NEXT J
140 NEXT I
```

PROGRAMAS PROPUESTOS

1. Escribir un programa que imprima la fecha en palabras a partir de la representación siguiente:

S, DD, MM, AA

en donde
S = Día de la semana, 1 a 7 (1 para LUNES, 2 para MARTES, etc.).
DD = Día del mes, 1 a 30 o 31, según los meses.
Fijar febrero con 28 días
AA = Dos últimas cifras del año.

Ejemplo: Se ha de introducir por programa cualquier fecha, 5, 24, 8, 83, y se ha de imprimir su fecha equivalente en palabras VIERNES 24 AGOSTO 1983.

2. Las notas de cinco estudiantes son:

	Geografía	*Matemáticas*	*Física*	*Química*
Alumno 1	7.4	6.8	4.2	3.6
Alumno 2	6.2	4.4	5	5
Alumno 3	4.8	3.2	2.8	1.6
Alumno 4	8.8	8.6	9.2	7.8
Alumno 5	6	5.2	6	7.3

Realizar un programa que lea la tabla (el fichero de nombres y notas están en sentencias DATA) de calificaciones. Calcule la media de cada estudiante de las cuatro asignaturas, la media por asignatura y la media total del grupo.
El formato de presentación de los datos de salida debe ser el anterior añadiendo una quinta columna. «NOTA MEDIA», una séptima fila «MEDIA DE ASIGNATURA» y una novena fila «MEDIA TOTAL DEL CURSO» con los valores obtenidos en cada caso.

3. Sea $f(x)$ definida por la ecuación:

$$f(x)=(x)^3 \cdot \text{sen}\,(x)$$

Realizar un programa que realice los siguientes apartados:

a) Calcular el valor de la función cuando x sea igual a 0, 0.5, 1.5, 2.

b) Calcular el valor máximo de los valores obtenidos.

c) Imprimir una tabla que represente los valores que se obtienen si x varía desde 0 a 2 en pasos de 0.1 en 0.1, con el formato siguiente:

x	x^3	sen x	$f(x)$
0	0	0	0
0.1	—	—	—
—	—	—	—

4. Escribir un programa que lea la siguiente matriz en sentencias DATA y después la imprima.

```
4  1  0  6  1
3  8  9  1  7
```

5. La tabla siguiente representa las ventas totales hechas por diferentes vendedores de una empresa en miles de pesetas a lo largo de la semana.

Vendedor	*LUN*	*MAR*	*MIE*	*JUE*	*VIE*	*SA*
1	35	120	70	50	72	90
2	64	80	60	80	70	110
3	72	130	40	100	74	90
4	90	140	30	95	76	120

Escriba un programa que calcule e imprima:

a) Ventas totales diarias.

b) Ventas totales semanales por vendedor.

c) Ventas totales semanales.

CAPITULO 9

Subrutinas-subprogramas

0. INTRODUCCION

Hasta ahora se han analizado numerosos conceptos del lenguaje BASIC, sin embargo, la noción de *subrutina* tal vez le confiera una gran potencia en la creación de programas coherentes y estructurados. Mediante subrutinas se podrá acortar la longitud de numerosos programas, así como ampliar la efectividad de los mismos.

Las subrutinas forman parte de cualquier programa, complejo o no, y tan importante es su presencia que existen gran cantidad de libros y paquetes de programas que son exclusivamente subrutinas de uso general y a los que los programadores recurren para ahorrar tiempo y trabajo.

Las subrutinas proporcionan un excelente método para organizar programas de computadora, de tal modo que sean más fáciles de entender, desglosándolos en secciones funcionales, que se pueden volver a utilizar cuando se desee.

A la terminación de este capítulo, podrá diseñar programas que tengan subrutinas incorporadas y usar las siguientes sentencias:

— GOSUB
— RETURN
— STOP
— ON GOSUB

1. ¿QUE ES UNA SUBRUTINA?

Una *subrutina, subprograma* o simplemente *rutina,* es una línea o líneas de instrucciones que se utilizan para ejecutar una tarea específica, muchas veces dentro de un programa, y a la que se la puede llamar en cualquier momento. Una subrutina termina siempre con una instrucción de retorno al programa que la llamó; puede ser llamada también por otra subrutina.

La llamada a subrutinas se realiza mediante las sentencias GOSUB y ON GOSUB con un número de línea que indica la línea de comienzo de la subrutina.

La ventaja de una subrutina es doble:

— Evita que se dupliquen secuencias de instrucciones idénticas que es preciso ejecutar varias veces en el curso de un programa.

— Permite reutilizar programas ya verificados y necesarios para otros programas.

Mediante subrutinas se puede realizar un nuevo método de construcción de programas que consiste en organizar un programa como módulos (subrutinas) ya *prefabricados.* Se puede

considerar a una subrutina como «una caja negra» en la cual se introducen unos datos y obtienen unos resultados sin preocuparse de lo que contenga su interior.

Las subrutinas permiten a los programadores encontrar más fácilmente los errores.

Una subrutina puede llamar a otra, pero el BASIC debe almacenar las direcciones de retorno intermedias y existe un límite en la imbricación o anidación; en general suelen permitirse hasta 16 subrutinas en cascada.

Al contrario que otros lenguajes de alto nivel, el BASIC no permite escribir subrutinas con independencia del programa que la llama, ya que un mismo nombre de variable designa la misma variable y la subrutina forma parte integrante del conjunto del programa.

2. SENTENCIA GOSUB-RETURN

La sentencia GOSUB (abreviatura de «GO to SUBroutine») significa literalmente «Ir a la subrutina» y está seguida por el número de línea en la que comienza la subrutina.

Así se tiene una subrutina que comienza en la línea 500, una instrucción que le llame sería GOSUB 500.

Una sentencia RETURN se debe utilizar al final de la ejecución de una subrutina para devolver el control desde la subrutina al programa principal.

Formato: *n GOSUB m* *n RETURN*

n Número de línea.
m Número de línea en que comienza la subrutina llamada.

Veamos una aplicación de una sentencia GOSUB para realizar una tarea repetitiva. Supongamos que a lo largo de un programa y en numerosas ocasiones se deben introducir números por teclado y han de ser precisamente enteros. La secuencia del programa sería:

```
......
100 INPUT "Introduzca un numero"; N
120 IF N<>INT (N) THEN 100
......
......
500 INPUT "Introduzca un numero"; N
520 IF N<>INT (N) THEN 500
......
700 INPUT "Introduzca un numero"; N
720 IF N<>INT (N) THEN 700
......
```

Evidentemente la repetición de las líneas 100, 120, 500, 520, etc., no sólo es tediosa sino que ocupa espacio de memoria que en muchas ocasiones será necesario. Imagínese que estas líneas repetitivas en vez de ser dos, fuesen decenas, centenas o a veces millares de líneas; la labor de introducción de las mismas y el espacio ocupado aumentarían considerablemente. En estas ocasiones es *necesario* recurrir a considerar la tarea repetitiva como subrutina y utilizar las sentencias GOSUB-RETURN.

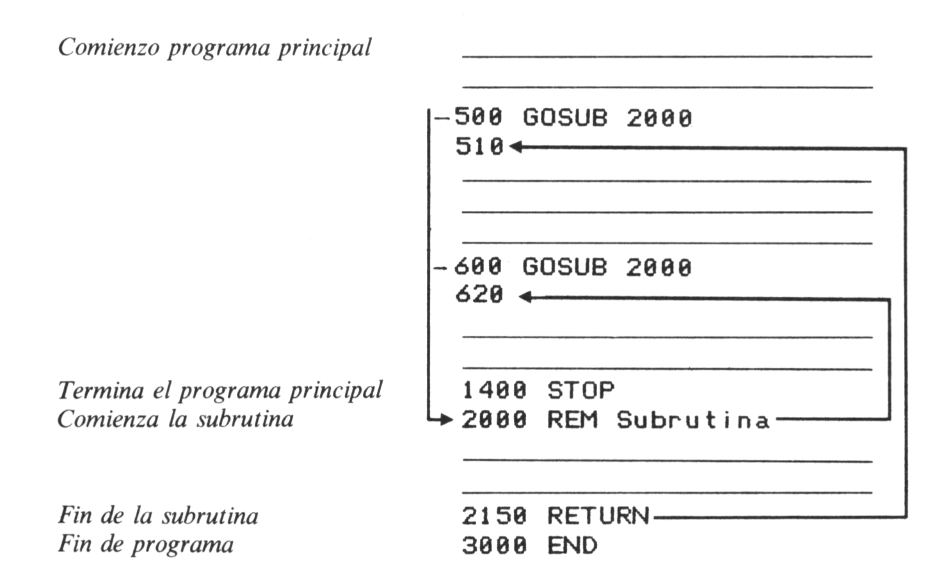
```
......
100 GOSUB 1000
120 ...
......
500 GOSUB 1000
520 ...
......
```

```
700 GOSUB 1000
720 ...
......
1000 INPUT "Introduzca un numero"; N
1020 IF N<>INT (N) THEN 1000
1030 RETURN
```

¿Qué sucede al ejecutar el programa? La línea 100 envía al control del programa a la subrutina que comienza en la línea 1000, se realizan las líneas 1000 y 1020, y al ejecutarse la línea 1030 la sentencia RETURN transfiere el control del programa a la línea siguiente (120 en nuestro caso). Al llegar el programa a la línea 500 se vuelve a saltar a la línea 1000, y así sucesivamente se producirán *llamadas* a las sentencias cada vez que aparezca GOSUB y se retorna a la línea siguiente de GOSUB cuando se ejecuta RETURN.

Como habrá observado el lector, la sentencia GOSUB es muy similar a GOTO, pero existe una diferencia esencial a favor de GOSUB, y es que ésta *recuerda* el número de la línea siguiente a la que debe volver cuando se termina la tarea repetitiva y se ejecuta RETURN. Por el contrario, GOTO puede realizar la tarea repetitiva pero cuando ésta se termina *el retorno exige otro* GOTO y conocer previamente el número de línea al que debe retornar.

DIAGRAMA DE USO DE SUBRUTINAS



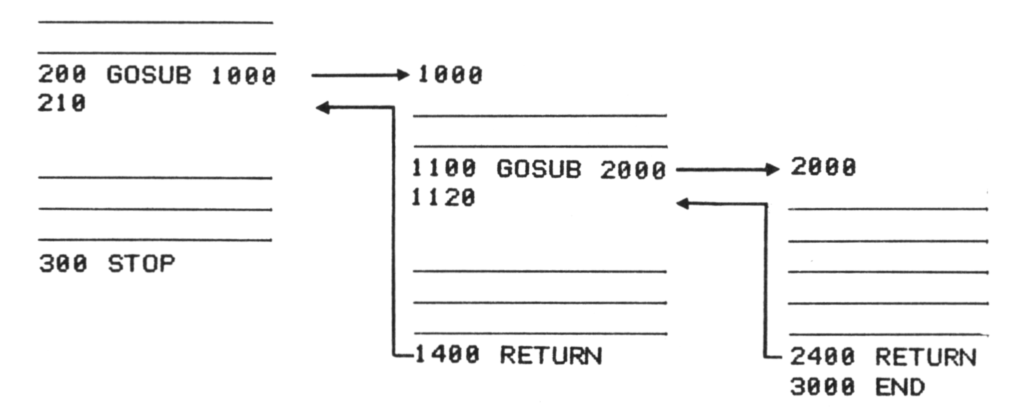
Si el programa anterior se ejecutara, cuando la computadora alcance GOSUB en la línea 500, el programa saltará al comienzo de la subrutina en la línea 2000. Se ejecuta la subrutina y cuando se encuentra RETURN en la línea 2150 el control pasará al siguiente número de línea más alto después del GOSUB de la línea 500; esto es, 510. Después la computadora sigue con el programa principal hasta llegar a la línea 600, de donde se bifurca de nuevo a la línea 2000. La sentencia RETURN salta esta vez a la línea 620.

Naturalmente, GOSUB 2000 se puede utilizar tantas veces se desee en el programa. Generalmente, la parte superior del programa es el programa principal, y las subrutinas se agrupan juntas al final. La razón de esta ubicación es que sólo se desea ejecutar las subrutinas cuando son llamadas por una sentencia GOSUB.

Normalmente, el programa principal se termina con una sentencia STOP (funciona igual que la orden STOP), a fin de evitar que cuando éste se termine se ejecute la subrutina sin ser llamada al efecto. La sentencia STOP interrumpe la ejecución del programa y equivale a unas sentencias GOTO y END, de tal forma que equivale a saltar a la línea END cuando se termina el programa principal.

En numerosas ocasiones es necesario saltar de una subrutina a otra, o lo que es igual «una subrutina puede llamar a otra».

Programa principal



El control del programa sería: 200, 1000 -- 1000, 2000 -- 2400, 1120 -- 2400, 210 -- 300, 3000.

De modo general, la sentencia RETURN reenvía el control a la línea siguiente al último GOSUB que se ha ejecutado.

Ejemplo de aplicación 1:

Supongamos que se desea obtener el área de un círculo o de un cuadrado cuyo lado/radio viene expresado en una lista contenida en una sentencia DATA.

Evidentemente, si el número de datos es grande, es conveniente utilizar dos subrutinas: una para cálculo del área del círculo ($\pi.r^2$) y otra para el área del cuadrado (l^2).

Area del círculo	A=3.141592*X^2	X=*radio círculo*
Area del cuadrado	A=X*X	X=*lado del cuadrado*

El programa puede ser el siguiente:

```
100 REM Areas cuadrado y circulo
120 PRINT "Menu"
140 PRINT "1-Circulo"
160 PRINT "2-Cuadrado"
180 INPUT H
200 IF H=1 THEN 250
220 IF H=2 THEN 350
240 IF H<1 OR H>2 THEN 120
250 READ X
255 IF X=999 THEN 3000
260 GOSUB 1000
280 PRINT "Area ="; A
300 GOTO 250
350 READ X
360 IF X=999 THEN 3000
400 GOSUB 2000
420 PRINT "Area ="; A
440 GOTO 350
500 DATA 4, 6, 5, 2, 10, 999
1000 A=3.141592*X^2
1020 RETURN
2000 A=X*X
2020 RETURN
3000 END
```

Ejemplo de aplicación 2:

Se desea calcular la media aritmética de cada dos números A, B introducidos por teclado.

Media aritmética $=\frac{A+B}{2}$

La subrutina «media aritmética» nos realizará el cálculo.

```
100 REM Media aritmetica
120 INPUT A, B
130 IF A=0 THEN 1200
140 GOSUB 1000
160 PRINT "La media de"; A; "y"; B; "es"; M
180 GOTO 120
200 REM Subrutina
1000 S=A+B
1010 M=S/2
1020 RETURN
1200 PRINT "Fin"
1300 END

RUN
? 4, 6
La media de 4 y 6 es 5
? 3, 9
La media de 3 y 9 es 6
? 0,0
FIN
```

3. NORMAS DE APLICACION DE SUBRUTINAS

Vamos a resumir a continuación las normas o reglas por las que se rige el tratamiento de subrutinas y que deberá tener en cuenta cuando trabaje con ellas:

1. Un programa puede tener cualquier número de subrutinas.
2. Una subrutina puede constar de cualquier número de líneas que contenga cualquier sentencia, orden o función válida en BASIC.
3. Las subrutinas se pueden situar en *cualquier posición* en un programa. Sin embargo, se suelen agrupar *al principio* o *al final* del programa, pero no en el medio del programa principal, a fin de facilitar la comprensión y legibilidad del mismo. Sin embargo, su ubicación más usual es al final del programa y *antes* de las líneas DATA si las hubiera. Una razón para ello es obtener mayor rendimiento en la velocidad del programa, ya que hay BASIC que exploran los números de línea en orden creciente para la sentencia GOTO y decreciente para GOSUB y en este caso interesaría interesar las subrutinas llamadas al final del programa. Por ello, para generalizar su emplazamiento se suelen situar casi siempre al final del programa.
4. Una subrutina puede ser ejecutada solamente por una sentencia GOSUB, que literalmente significa «Ir a la subrutina» (igual que a una línea DATA únicamente se puede acceder por una sentencia READ).
5. El final de una subrutina es siempre una sentencia RETURN que significa «Retorno al punto o línea *después* de la sentencia GOSUB que la llamó, y a continuación su ejecución a partir de la mencionada línea».
6. La parte principal de un programa está separada de la/s subrutina/s mediante una sentencia STOP o END que significa «Este es el final del programa principal; cuando se alcance esta línea, *deténgase*». Normalmente, se suele utilizar STOP dado que END se reserva para la línea última del programa.
7. Las subrutinas pueden ser llamadas desde el interior de otras subrutinas. El número máximo de subrutinas en cascada suele variar de unas computadoras a otras, siendo 16 un número medio. Las direcciones de retorno de las diferentes subrutinas se almacenan en una memoria del tipo «pila» de modo tal que los retornos se hacen en orden inverso a las llamadas.

4. SENTENCIA ON-GOSUB

La sentencia ON-GOSUB realiza la misma función que ON-GOTO respecto a subrutinas.

Formato: | *n* ON *variables* GOSUB *n.º de línea*$_1$, *n.º de línea*$_2$,...

n	Número de línea.
variable	Variable o expresión válida.
n.º de línea	Número de línea a la que se bifurca.

Según el valor de la variable o expresión, se realiza la ejecución de una de las subrutinas especificadas y después retorna a la línea siguiente a ON-GOSUB.

Si *variable*=1 --- se bifurca al número de línea 1 (*n.º de línea*$_1$)
Si *variable*=2 --- se bifurca al número de línea 2 (*n.º de línea*$_2$)
Si *variable*=3 --- se bifurca al número de línea 3 (*n.º de línea*$_3$)
....

```
100 ON K GOSUB 1000,2000,3600
```

Si K vale 1 se bifurca a la subrutina que comienza en 1000. Si K vale 2 se bifurca a la subrutina que comienza en 2000. Si K vale 3 se bifurca a la subrutina que comienza en 3000. Si el valor de K es menor que 1 o mayor de 3 no se bifurca a ninguna subrutina, dependiendo de cada computadora el que se produzca un mensaje de error o se salte a ejecutar la línea siguiente de programa.

```
100 ON K GOSUB 1000,2000,3600
.
.
.
```

```
1000
.
.
1270 RETURN
```

Si K es 1, se hace esta subrutina.

```
.
.
```

```
2000
.
.
2490 RETURN
```

Si K es 2, se hace esta subrutina.

```
.
.
```

```
3600
.
.
3990 RETURN
```

Si K es 3 se hace esta subrutina.

Ejemplo:

```
10 REM Demo ON-GOSUB
20 PRINT "Introduzca opcion 1-2-3"
30 INPUT H
40 ON H GOSUB 500, 600, 700
50 GOTO 20
500 REM Subrutina 1
520 PRINT "Bifurcacion a subrutina 1"
540 RETURN
600 REM Subrutina 2
620 PRINT "Bifurcacion a subrutina 2"
640 RETURN
700 REM Subrutina 3
720 PRINT "Bifurcacion a subrutina 3"
740 RETURN

RUN
Introduzca opcion 1-2-3
? 2
Bifurcacion a subrutina 2
Introduzca opcion 1-2-3
? 1
Bifurcacion a subrutina 1
Introduzca opcion 1-2-3
? 3.546
Bifurcacion a subrutina 3
.........................
```

4.1. Aplicación de ON-GOSUB. Los menús

En el Capítulo 4 ya se introdujo el concepto de *menú*, lista de opciones, y el procedimiento usual de visualización y selección mediante las sentencias IF-THEN y IF-GOTO. En este párrafo se vuelve a introducir el concepto de menú, cómo visualizarlo en la pantalla, y el método que ha de seguir el usuario para la selección de la opción deseada, tecleando un número, todo ello mediante la sentencia ON-GOSUB.

El proceso a seguir es el siguiente:

a) Visualizar el menú.
b) Seleccionar el número que indica la opción deseada.
c) Ejecutar la subrutina adecuada.

Cada opción listada en el menú puede corresponder a un bloque de pasos en una subrutina. Aunque el ejemplo ya visto al principio del párrafo 4 es ilustrado, vamos a ver un programa más completo que incluya la visualización de un menú, su selección y las correspondientes subrutinas. Supongamos que el programa puede contener cuatro subrutinas, con el siguiente menú:

```
ELIJA UNA OPCION MEDIANTE UN NUMERO
1- Introducir nuevos datos
2- Modificar datos
3- Visualizar datos actuales
4- Terminar
```

```
100 N=0
110 WHILE L<5
120 GOSUB 10000  'menu principal
```

```
130 INPUT "Seleccione opcion";N
140 ON N GOSUB 11000,12000,13000,14000
150 PRINT "Seleccione correctamente"
160 GOTO 130
170 WEND
180 PRINT
199 END
10000 'Subrutina menu principal
10100 PRINT "ELIJA UNA OPCION MEDIANTE UN NUMERO"
10200 PRINT " 1-Introducir nuevos datos"
10300 PRINT " 2-Modificar datos"
10400 PRINT " 3-Visualizar datos actuales"
10500 PRINT " 4-Terminar"
10600 RETURN
```

```
11000 'Subrutina de introducir datos
.
.                       '(sentencias subrutinas)
.
11890 RETURN
```

```
12000 'Subrutina de modificar datos
.
.
12960 RETURN
```

```
13000 ' Subrutina de visualizar datos actuales
.
.
13890 RETURN
```

```
14000 'Subrutina de terminar programa
14900 RETURN
```

PROGRAMAS RESUELTOS

PROGRAMA 1: *Obtener la función X = SQR(A)+LOG(B) de una serie de parejas de datos contenidos en sentencias DATA.*

```
100 REM Demo GOSUB
105 PRINT " A", " B", " X"
110 READ A, B
120 IF A=0 THEN 200
130 GOSUB 160
140 PRINT A, B, X
150 GOTO 110
160 REM Subrutina
170 X=SQR (A)+LOG (B)
180 RETURN
190 DATA 4, 5, 7, 1, 8, 14, 9, 13, 0, 0
200 END

RUN
 A                B                X
 4                5                3.60944
 7                1                2.64575
 8                14               5.46749
 9                13               5.56495
```

PROGRAMA 2: *Subrutinas anidadas.*

```
100 REM Demo
110 GOSUB 200
120 PRINT " A", " B", " X"
130 PRINT A, B, X
140 STOP
200 REM Subrutina 1
210 INPUT "A, B"; A,B
220 GOSUB 300
230 X=X*3
240 RETURN
300 REM Subrutina 2
310 X=(A*B)/2
320 RETURN
400 END

RUN
A, B? 16, 4
 A                B                X
 16               4                96
RUN
A, B? 4, 1
 A                B                X
 4                1                6
```

PROGRAMA 3: *Subrutinas sucesivas.*

```
100 PRINT "Ingrese numero"
110 INPUT N
120 IF N<0 THEN 100
```

```
130 GOSUB 400
140 GOTO 440
400 REM Suma de 1 a N
410 S=(N * (N+1)) / 2
420 PRINT "Suma ="; S
430 RETURN
440 N=N*2
450 PRINT "N*2 ="; N
460 GOSUB 400
999 END

RUN
Ingrese numero
? 5
Suma = 15
N*2 = 10
Suma = 55
```

PROGRAMA 4: *Subrutinas anidadas.*

```
10 PRINT "Introducir numero"
20 INPUT N
30 IF N<0 THEN 20
40 GOSUB 100
50 STOP
100 REM Suma de 1 a N
110 S1=(N * (N+1)) / 2
120 PRINT "Suma ="; S1
130 PRINT "Suma de los cuadrados S/N";
140 INPUT A$
150 IF A$="S" THEN GOSUB 200
160 RETURN
200 REM Suma cuadrado
201 S2=(N * (N+1) * (2*N+1)) / 6
202 PRINT "Suma de los cuadrados S ="; S2
203 RETURN

RUN
Introducir numero
? 5
Suma = 15
Suma de los cuadrados S/N? S
Suma de los cuadrados S = 55
```

PROGRAMA 5: *Aplicación de* ON GOSUB.

```
5 X=7
10 FOR I=1 TO 3
20 ON I GOSUB 200, 300, 400
30 NEXT I
100 STOP
200 PRINT X, SIN (X)
210 RETURN
300 PRINT X, X*12, COS (X)
310 RETURN
400 PRINT X, X*13, TAN (X)
410 RETURN

RUN
 7              .656987
 7              84              .753903
 7              91              .871448
```

PREGUNTAS Y PROGRAMAS DE REPASO

1. ¿Qué error se ha cometido en este programa?

```
100 PRINT "Introduzca un numero"
110 INPUT N
120 IF N<INT (N) THEN 110
130 RETURN
140 GOSUB 100
200 END
```

2. ¿Qué diferencia existe entre ON-GOTO y ON-GOSUB?

3. Estudiar el siguiente programa:

```
10 INPUT A
20 IF A=0 GOTO 200
30 GOSUB 100
40 GOTO 10
100 B=A^3
110 PRINT A, B
120 RETURN
200 END
```

¿Qué misión cumple la línea 40?
¿Qué sucedería si no existiera?

4. Examine el siguiente programa:

```
10 INPUT "Introduzca 4 numeros"; A, B, C, D
20 GOSUB 190
30 PRINT A, B, E
40 GOSUB 300
50 PRINT C, D, E
60 GOTO 400
190 REM Subrutina 1
200 IF A>B THEN 230
210 E=A*B
220 RETURN
230 E=A/B
240 RETURN
300 REM Subrutina 2
310 IF C>D THEN 340
320 E=C*D
330 RETURN
340 E=C/D
350 RETURN
400 END
```

Ejecute el programa y describa las operaciones que realiza.

5. Escribir un programa que permita seleccionar a voluntad una cualquiera de estas subrutinas:

   a) Leer e imprimir tres variables introducidas por teclado y su producto.

   b) Sumar e imprimir cuatro números leídos de sentencias DATA.

   c) Obtener la raíz cuadrada y el logaritmo de los números introducidos por teclado.

6. Escribir un programa que permita seleccionar a voluntad una cualquiera de estas subrutinas:

   a) Creación de un fichero.

   b) Lectura de un fichero.

   c) Modificación de los elementos de un fichero.

   d) Introducción de nuevos elementos.

   e) Borrado de elementos.

   f) Fin.

PROGRAMAS PROPUESTOS

1. Se desea escribir un programa para estadística que permita obtener la media aritmética, varianza y desviación típica de una muestra de población de acuerdo con las siguientes condiciones:

   — El número de individuos (N) se introducirá por teclado.

   — Se elegirá entre los parámetros PESO/ESTATURA del individuo (*aplicación* de ON-GOSUB).

   — Una vez elegido el parámetro, se introducirán por teclado los N valores correspondientes de la muestra.

   — Una subrutina calculará la media aritmética (M)/varianza (V)/desviación típica (E).

   — Los resultados presentarán: Número de individuos (N), parámetro elegido, valores del parámetro en los N individuos y media aritmética, varianza y desviación típica.

NOTA: *Se recuerda que si son N individuos y X_i la medida o valor relativo a cada individuo, los valores de M, V y E son:*

$$\text{Media aritmética, M} = \frac{\sum_{i=1}^{n} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_n}{N}$$

$$\text{Varianza, V} = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{M})^2$$

$$\text{Desviación típica, E} = \sqrt{V}$$

2. Se necesita un programa que realice el control de un inventario con las siguientes funciones:

   — Modificación del precio de coste de cada artículo.

   — Actualización de la cantidad de artículos en *stock.*

   — Verificación del estado actual de cada artículo.

   — Impresión del inventario completo.

   — Salida del programa.

CAPITULO 10

Sentencias-funciones y órdenes específicas

0. INTRODUCCION

El lenguaje BASIC es, sin duda, el que posee más variaciones en las ampliaciones o extensiones que ofrecen las diferentes versiones y computadoras. Aunque no se pueden resumir todas las variaciones en unas páginas, vamos a tratar todas aquellas sentencias/funciones y órdenes específicas que son más utilizadas en la actualidad.

En este capítulo nos limitaremos a todas las sentencias/órdenes (comandos) que se encuentren en la mayoría de los sistemas y que tengan gran utilidad.

Distinguiremos tres grandes grupos:

— *Ordenes básicas del sistema.*
— *Ordenes de depuración del programa.*
— *Funciones específicas o especiales.*

Así, a la terminación de este capítulo el lector sabrá manejar las órdenes básicas del sistema:

AUTO/RENUMBER-REN EDIT/DELETE CLEAR/CONT/STOP/NEW	*Ordenes de utilidad y de edición*

Como estas órdenes de utilidad ya se han comentado en los Capítulos 1 y 2, remitimos al lector a los mismos sugiriéndoles una nueva lectura.

Las funciones o sentencias especiales:

PEEK-POKE	*Acceso a memoria.*
CALL-USR-SYS	*Llamadas a subrutinas en lenguaje máquina.*
WAIT-TIME$-FRE-MEM	*Especiales.*
SWAP, CLEAR, CLS, IN-OUT	*Acceso a ports de entrada-salida (E/S).*
	Borrado de memoria y pantalla.

Ordenes o sentencias de depuración:

ON ERROR GOTO	*Detección de errores*
ERROR, ERR, ERL	*Rastreo de errores*
TRON - TROF (TRACE / NO TRACE)	
DSP (DISP)	

1. SENTENCIAS-FUNCIONES PEEK-POKE

PEEK se suele considerar una función y POKE una sentencia. Permiten el control directo de posiciones de memoria a partir de un programa. Mediante ellas se podrán realizar un número considerable de aplicaciones, entre las que podemos destacar: control de dispositivos externos a la computadora, paso a subrutinas en lenguaje máquina, control de la información en pantalla mediante gráficos y generación de colores, control de sonidos, etcétera.

Antes de proseguir, recordemos que un octeto (byte) es una palabra digital de 8 bits y que una posición de memoria (en las microcomputadoras de 8 bits) ocupa un octeto. Dado que el bus de direcciones tiene 16 hilos, se podrá tener como máximo 2^{16} posiciones (65.536), por lo que un número próximo a éste serán las posiciones libres de memoria. Como un octeto tiene 8 bits se podrán representar $2^8=256$ palabras distintas en una posición de memoria, o lo que es igual en una posición de memoria cualquiera se podrán escribir hasta 256 palabras distintas o lo que es igual los equivalentes binarios de los números decimales 0, 1, 2,..., 255.

La ejecución y escritura directa en memoria son las posibilidades más interesantes y le permitirán comunicarse con programas escritos en lenguaje máquina e incluso con periféricos que ocupan una posición de memoria en el «mapa de memoria». Para ello se «ve» lo que hay en la memoria y en función de su contenido se toma la acción correspondiente.

1.1. Sentencia POKE

POKE se utiliza para almacenar valores enteros desde 0 a 255 (en decimal) en posiciones de memoria específicas. Por ejemplo POKE 13425,66 sitúa el código ASCII del número 66 (letra B) en la dirección de memoria 13.425.

Formato: POKE I,J

I, expresión variable o constante que indica la dirección de la memoria entre 0 y la posición más alta de la misma (63999 en Apple II y 65535 en PET/CBM).

J, número entero 0 y 255

A cada posición de la memoria se le asigna un contenido. Se dice que la sentencia POKE *escribe* en la memoria:

```
POKE 32768, 140
```

El valor 140, o su equivalente en código ASCII, se almacena en la dirección 32768, borrándose el contenido que tuviera anteriormente.

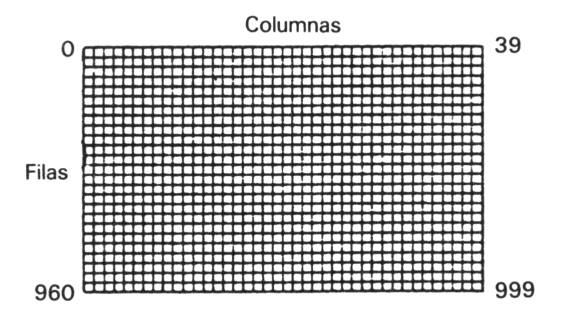
```
100 FOR I=32768 TO 32852
110 POKE I, 66
120 NEXT I
```

En las direcciones 32768 a 32852 de memoria se almacena el carácter correspondiente al código ASCII de 66 (letra B).

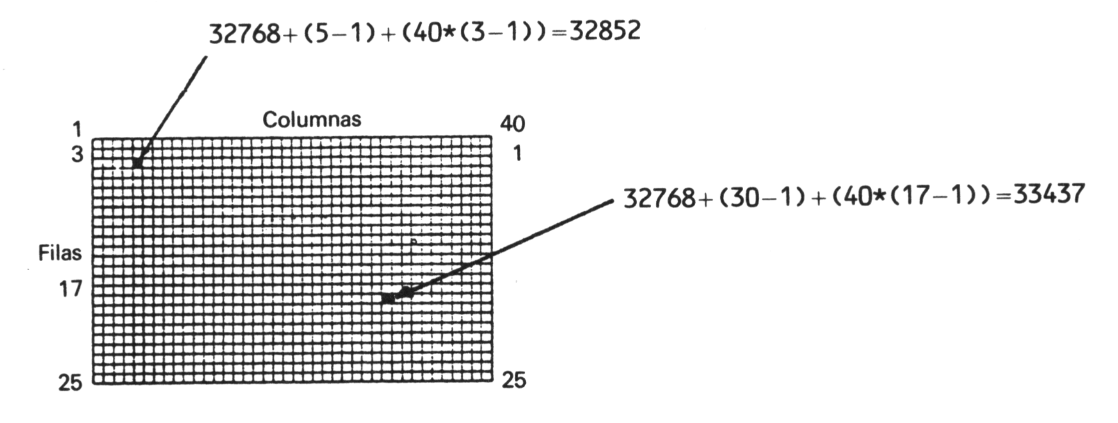
La memoria de una computadora se asemeja a un conjunto de celdillas de la siguiente forma:

Posición o dirección 0	______	*Contenido de la posición 0*
Posición o dirección 1	______	.
Posición o dirección 2	______	.
Posición o dirección 3	______	.
Posición o dirección .	______	.
Posición o dirección .	______	.
	______	
Posición o dirección 32768	66	*Código ASCII de 66, letra B*
.	______	
.	______	
.		
.	______	
Posición o dirección 63999	______	

Como ejemplo consideremos la pantalla de una computadora Commodore 64 (25 filas ×40 columnas=1000 posiciones de memoria). Se puede considerar la pantalla como una retícula de 1000 cuadrados organizados en 25 filas y 40 columnas.



Se puede visualizar un carácter en cada cuadrado. A cada posición de la pantalla se le asigna una dirección en la memoria y un espacio de un octeto. La pantalla se relaciona con la memoria, color de los caracteres que se representan en la memoria, pantalla y que asocia el cuadrado superior izquierda con la dirección 55296 y el cuadrado inferior derecha, como se ve en la siguiente retícula, en la dirección 56295.



Así, por ejemplo, escriba:

```
100 POKE 33188, 42
```

Pulse RUN y RETURN. El símbolo asterisco (*) en color se dibujará en el centro de la pantalla (en el cuadrado asociado a la dirección 55716). Si desea saber qué posición corresponde a un cuadrado específico, cuente la fila F y columna C y aplique la fórmula:

```
32768+(C-1)+(40*(F-1))
```

Un cuadrado situado en la fila 4, columna 5, corresponde a la posición de memoria: 55296+(5−1)+(40*(4−1))=32768+4+120=55416. Mediante esta ecuación se pueden ejecutar sentencias POKE de caracteres en la pantalla.

Con POKE se puede *modificar* cualquier posición de memoria lectura-escritura (RAM). Una sentencia POKE ejecutada a una dirección correspondiente de memoria de sólo lectura (ROM o PROM) no tiene ningún efecto. Sin embargo, debe tener cuidado en este último caso ya que una sentencia POKE sobre una zona de memoria ROM, PROM puede hacer perder el control de la computadora.

1.2. Función PEEK

La función PEEK se utiliza para examinar el contenido de una posición o dirección específica en la memoria de la computadora. Por ejemplo, X=PEEK (32768) asigna el valor numérico almacenado en la dirección de memoria 32768 a la variable X.

La sentencia PEEK devuelve el contenido de una dirección de memoria con un número comprendido entre 0 y 255. PEEK se puede utilizar con la sentencia POKE para leer el contenido de la posición de memoria que previamente ha sido escrita por ella.

Formato: | PEEK *(dirección de memoria)* |

dirección de memoria: en el rango 0 a 65535

Por ejemplo:

```
100 K=PEEK(346)
```

Situará en la variable K el contenido de la dirección de memoria 346. El contenido de la dirección 364 no se altera.

La función PEEK devuelve un entero de 0 a 255 que representa el contenido del octeto de memoria de la dirección indicada. Mediante PEEK se puede examinar una posición de memoria (por ejemplo en dibujos animados para detectar un «choque» entre objetos y así programar un rebote), detectar la pulsación de una tecla, o se pueden consultar determinadas tablas o subrutinas en lenguaje máquina de memoria, miniensamblador, etc.

Se puede *realizar* una sentencia PEEK en memoria de lectura escritura (RAM) o en memoria de sólo lectura (ROM). A diferencia de POK, que puede producir daños en la memoria, PEEK es inofensiva.

1.3. Aplicaciones de PEEK-POKE

PEEK y POKE existen en la mayoría de las microcomputadoras o computadoras personales, pero al ser delicado su uso no suelen estar presentes en los BASIC de los sistemas grandes.

En este apartado analizamos algunas de las aplicaciones más usuales de PEEK/POKE.

1.3.1. Tratamiento de la información en pantalla

Las computadoras que poseen una zona específica de memoria, llamada «memoria pantalla», permiten mediante PEEK/POKE la representación de gráficos y de caracteres en la pantalla sin necesidad de recurrir a las sentencias PRINT.

Estas computadoras están organizadas en formato de «mapa de memoria» de modo que se exploran de forma constante y automática todas las posiciones de memoria. En las computadoras Commodore la pantalla de 25 filas × 40 columnas ocupa 1000 posiciones de memoria o cuadrados pequeños, de modo que el espacio de memoria pantalla comienza en la dirección 32768 para el cuadrado 1 (fila 1, columna 1), y finaliza en la dirección 33767 para el cuadrado 1000 (fila 25, columna 40).

Ejemplo a): El programa siguiente rellena una fila de la pantalla con letras A (código 65 en ASCII) al ejecutarse.

```
100 FOR A=33088 TO 33127
110 POKE A, 65
130 NEXT A
```

Como se visualiza la línea muy rápidamente, ralentice la presentación mediante un bucle vacío:

```
120 FOR B=1 TO 100: NEXT B
```

Ejemplo b): Se pueden obtener los 256 símbolos del conjunto completo correspondiente a una computadora mediante un bucle de la forma siguiente:

```
120 FOR N=0 TO 255
140 POKE 33267, N
160 NEXT N
```

Al ejecutar este programa se imprimen cada uno de los 256 símbolos que se encuentran en el teclado en el centro de la pantalla (posición 55296). Dado que esta presentación será muy rápida, se ralentiza mediante la línea 150.

```
150 FOR J=1 TO 400: NEXT J
```

NOTA: *Las direcciones de memoria son específicas de cada computadora. Si desea realizar los ejemplos* a *y* b *anteriores con su computadora deberá consultar en su Manual las direcciones de la memoria pantalla. Así, por ejemplo, en las viejas computadoras:*

DRAGON 32	Direcciones memoria pantalla 1024-1535.
VIC-20	Direcciones memoria pantalla 38400-38905.
APPLE II	Direcciones memoria pantalla 1024-21208.

En el ejemplo *a*, para el DRAGON-32 cambie la línea 100 por:

```
100 FOR A=1024 TO 1535
```

En el ejemplo *b*, para el VIC-20 cambie la línea 140 por:

```
140 POKE 38647, N
```

Ejemplo c): Si desea hacer un *volcado* de memoria (visualizar los contenidos de una zona de memoria) en las direcciones 1000 a 1200, realice el siguiente programa:

```
100 FOR I=1000 TO 1200
120 A=PEEK (I)
140 PRINT I, A
160 NEXT I
```

Se imprimirá el número de la dirección y su contenido.

1.3.2. Establecimiento de opciones y controles de la computadora

PEEK y POKE se utilizan en muchas computadoras para definir diversas modalidades de funcionamiento, así como obtener el estado de diferentes funciones.

Por ejemplo, en las computadoras PET/CBM existen dos juegos de caracteres: estándar y alternativo. El juego estándar escribe siempre con mayúsculas y el alternativo escribe igual que una máquina de escribir. No están disponibles directamente y se accede a ellos mediante el uso de POKE. Al encender la computadora se trabaja en modo estándar; si se escribe

```
POKE 59468, 12
```
<RETURN>

se pasa al juego de caracteres alternativos. Para pasar al juego de caracteres estándares escriba

```
POKE 59468, 14
```
<RETURN>

Otro ejemplo podría ser el empleo del altavoz mediante programación con la microcomputadora VIC-20. Las siguientes sentencias controlan el altavoz:

Ordenes-Sentencias de voz	*Valor de X*	*Función*
`POKE 36878, X`	0al 15	ajusta volumen
`POKE 36874, X`	128 al 255	ajusta tono
`POKE 36875, X`	128 al 255	ajusta tono
`POKE 36876, X`	128 al 255	ajusta tono
`POKE 36877, X`	128 al 255	ajusta «ruido»

La mencionada VIC-20 cambia los colores del fondo y marco de la pantalla del monitor con:

```
POKE 36879, X
```
X es un número contenido en una tabla que da el fabricante.

1.3.3. Control de dispositivos externos

Tal vez una de las aplicaciones más importantes de PEEK/POKE sea el control de dispositivos externos de hardware conectados a la computadora. Estos dispositivos pueden ir desde palancas de control para juegos, sensores, etc., hasta sofisticados diseños electrónicos.

Para emplear PEEK y POKE es preciso considerar el tipo de circuito que se va a conectar, así como el interface existente entre la computadora y el dispositivo a controlar.

Para la realización del control del circuito se considera éste como si fuera una posición de memoria, de tal forma que mediante PEEK se leerá la información producida en el circuito y POKE permitirá enviar información al circuito.

Las computadoras deben poseer conectores que permitan los interfaces adecuados. Entre los más usuales se encuentran:

— *El puerto* (port) *del usuario.* Es un conector que permite disponer de 8 bits en paralelo (en micros de 8 bits), y es el más utilizado para control mediante programa.
— *Conector de magnetófono a casete.* Se utiliza para conectar ambos.
— *Conector IEEE 488.* Interfaz paralelo estándar de especificaciones conocidas; permite la conexión de gran número de aparatos, entre ellos instrumentación.
— *Ampliación de memoria.* Conector o ranura que permite la ampliación de la memoria central de la computadora.

Si nos ceñimos al puerto del usuario, que es el más fácil de acceder por el programador, y tomamos como ejemplo el VIC-20, la dirección del puerto es 37136. La tensión del pin (patilla) correspondiente del conector será 5 voltios si se escribe un 1. De igual forma si se lee la direc-

ción 37136 mediante PEEK se obtiene un 1 si el pin está puesto a 5 voltios por el exterior, o un 0 si se pone a 0 voltios.

```
POKE 37136, 128
```

como $128=2^7$ y su equivalente binario 10000000, significa que el bit 8 será un 1 y, por consiguiente, en el pin del conector correspondiente al bit 8 habrá un «1» de tensión, o sea 5V, que se podrá utilizar para activar, por ejemplo, un relé.

En resumen, en el port del usuario habrá siempre un número binario de 8 octetos que se podrá tratar en el exterior considerando los «1», 5V y los «0», 0V, o bien una información procedente del exterior en 8 hilos como 5 ó 0V, se tratará como 0 y 1 (información binaria para la computadora).

Si el conector de salida se conectara directamente a un altavoz, por ejemplo, haciendo variar el índice de POKE, o repitiendo un número determinado de veces la misma sentencia POKE, se pueden conseguir sonidos en el altavoz.

```
100 FOR I=1 TO 200: POKE 37136,1: NEXT I
```

2. FUNCIONES INP-OUT

Algunas microcomputadoras basadas en los microprocesadores 8085 y Z80 disponen de posiciones específicas de entrada y salida, denominadas ports (puertas) de E/S (entrada/salida), mediante las cuales se proporcionan un método para el intercambio de datos entre el microcomputador y su mundo exterior (periféricos) en ambos sentidos.

Este método que permite direccionar los canales de E/S se diferencia de PEEK/POKE en que no utilizan el mapa de memoria aunque se puede considerar una *memoria de puertos* que tiene 256 direcciones que se pueden decodificar fácilmente y asignarlas a los diversos periféricos externos, tales como teclados, relés, DAC (Convertidor Digital-Analógico), ADC (Convertidor Analógico-Digtal), etc.

Se accede a estos 256 ports mediante las funciones INP y OUT (abreviaturas de INPUT y OUTPUT) y se considera que estos 256 puertos de E/S pertenecen a un espacio especial (0-255), pero no a la memoria de la unidad central. Mediante un puerto se realizará el intercambio de datos de 8 octetos entre la computadora y los circuitos exteriores en ambos sentidos.

Las microcomputadoras TRS-80, IBM PC, MSX, Amstrad (en general todas las basadas en microprocesadores Z-80 y 8086) poseen estas funciones.

2.1. INP

INP proviene de «INput from a Port» (Entrada desde un puerto *).

Formato: | *n* INP (X) |

n Número de línea.
X Número de puerto.

La función INP se utiliza para leer el valor decimal de un octeto (X) de información en un puerto específico de la computadora. El valor del octeto está comprendido entre 0 y 255.

Por ejemplo:

```
10 PRINT INP (X)
```

que determina el puerto correspondiente X.

* *Puerto:* Es una vía de acceso al microprocesador desde el exterior.

INP es una herramienta útil para chequear los puertos para una condición específica tal como una petición de entrada desde un dispositivo periférico. Otras aplicaciones pueden incluir lecturas de temperaturas desde sensores remotos, presiones, etc.

Programa ejemplo;

```
100 REM Programa test "INP"
110 FOR X=0 TO 255
120 PRINT "Valor decimal del octeto en el port num."
130 PRINT X; "es"; INP (X)
140 NEXT X
200 END

RUN
Valor decimal del octeto en el port num.
0 es 255

......................

Valor decimal del octeto en el port num.
255 es 127
```

2.2. OUT

La función OUT se utiliza para enviar el dato especificado en decimal (valor del octeto) a un puerto de salida especificado del computador.

Formato: | *n* OUT *m, d* |

n	Número de línea.
m	Número de puerto.
d	Dato (octeto a enviar).

Los valores del octeto y del puerto deben ser enteros o variables positivas entre 0 y 255. Por ejemplo: OUT 254, 4 envía el equivalente binario de 4 (decimal) al número del puerto 255.

Programa ejemplo:

```
100 REM Programa test "OUT"
110 FOR I=0 TO 255
120 FOR J=0 TO 255
130 PRINT "Port ="; I
140 PRINT "Octeto ="; J
145 OUT I, J
150 NEXT J
160 NEXT I
999 END
```

En la microcomputadora TRS-80, si se envía el octeto 4 mediante OUT se pone en marcha el casete que conserva datos y programas, y si se envía el octeto 0 se detiene el casete.

```
100 REM Activacion cassette
110 OUT 255,4
120 OUT 255,0                      Puesta en marcha casete.
200 END                            Detención del casete.
```

Una de las aplicaciones de OUT es afectar a la salida de vídeo. En algunas pantallas unidas al Adaptador de Monitor Color/Gráficos, en el caso de la computadora IBM PC, se puede encontrar que los primeros dos o tres caracteres en la línea aparecen en la pantalla. Si su pantalla no tiene un control de ajuste horizontal, puede utilizar las siguientes sentencias para desplazar la visualización:

```
OUT 980,2: OUT 981,3:
```

mueve la visualización de dos caracteres a la derecha en el caso de que la anchura sea de 40 columnas (o 16 puntos en la modalidad de gráficos de media resolución, o 32 puntos en la modalidad de gráficos de alta resolución).

```
OUT 980,2: OUT 981,85
```

mueve la visualización cinco caracteres a la derecha en el caso de que la anchura sea de 80 columnas.

El movimiento provocado por listas sentencias OUT permanece en efecto hasta que se ejecuta una sentencia WIDTH o SCREEN. La orden MODE del DOS realiza los mismos efectos que las sentencias anteriores, con la ventaja de permanecer en funcionamiento hasta que se ejecute una Reinicialización del Sistema.

3. FUNCIONES DE LLAMADA A SUBRUTINAS EN LENGUAJE MAQUINA

La programación en lenguaje máquina sobrepasa los objetivos de este libro. Dicha programación es un poco más difícil que en lenguaje BASIC y depende estrechamente del microprocesador en que está basada la microcomputadora.

Sin embargo, gracias a los programas escritos en lenguaje máquina se pueden efectuar operaciones que sería muy difícil de efectuar sólo en BASIC. De igual forma ciertas funciones, pero sobre todo ciertas velocidades de ejecución, no son posibles más que en lenguaje máquina.

La escritura de un programa en lenguaje máquina permitirá obtener programas potentes y veloces. Aunque no conozca la programación en lenguaje máquina podrá utilizar sus características fundamentales basándose en sentencias que permiten la llamada a subrutinas en lenguaje máquina.

Un programa o subrutina en lenguaje máquina reside en una zona de memoria que no está ocupada por el BASIC, tiene una cierta dirección de comienzo en el espacio de memoria libre y se considera como una especie de subrutina.

Las subrutinas en lenguaje máquina son llamadas mediante las sentencias CALL, USR y SYS en los programas en BASIC; el retorno a la terminación se produce mediante la instrucción RTS o RET.

3.1. Sentencia CALL

La sentencia CALL transfiere el control del programa a una subrutina en lenguaje máquina que tiene su posición de entrada en la posición de memoria especificada.

Formato: *n* CALL (X)

n Número de línea.
X Posición de comienzo de la subrutina.

La subrutina puede ser parte del software interno de la computadora o puede ser escrita por el usuario. Los programas en lenguaje máquina escritos por el usuario se pueden introducir mediante sentencias POKE desde el teclado, mediante un programa BASIC o utilizando un programa «monitor/editor» a nivel de sistema.

Ejemplo: 100 CALL 16846 bifurcará a una subrutina escrita en lenguaje máquina almacenada a partir de la posición decimal 16846. Cuando una instrucción de retorno (RTS o RET) se encuentra en la subrutina, se vuelve al BASIC y se ejecuta la sentencia siguiente a la línea 100 que contiene a CALL, y a continuación el resto del programa.

Así pues, la forma de la sentencia CALL es similar a la de GOSUB.

Como se ve, puede emplearse CALL sin necesidad de escribir o saber lenguaje máquina, aunque evidentemente un buen programador debe conocer este lenguaje por su gran aplicación, aunque eso requiera estudiar el lenguaje ensamblador del microprocesador interno.

Normalmente el fabricante suele dar las direcciones de comienzo de diferentes subrutinas de utilidad a las que se llega mediante CALL desde el programa en BASIC. Las llamadas y retornos a las subrutinas internas se realizan en numerosas ocasiones sin que el programador haga nada al respecto; no obstante, en otras será preciso recurrir a la sentencia CALL.

3.2. Función USR

La función USR permite llamar a una subrutina en lenguaje máquina y posteriormente continuar con la ejecución del programa en BASIC.

Formato: | *n* USR (X) |

n Número de línea.
X Dirección de comienzo de la subrutina.

La subrutina en lenguaje máquina se puede introducir en memoria desde el teclado utilizando la sentencia POKE, o desde la unidad de cinta o disco utilizando la orden SYS.

Esta función es similar a CALL y se puede utilizar en programas de igual modo que si fuera una función incorporada del sistema.

Ejemplo:

```
100 PRINT USR (X)
```

Imprimirá el valor que se calcula mediante la subrutina del usuario en lenguaje máquina que comienza en X; si esta subrutina calculase la raíz cuadrada de X, la línea 100 imprimirá la raíz cuadrada de X.

Para comprobar la función USR, debe cargar una subrutina en lenguaje máquina en la computadora (en su dirección específica) utilizando la sentencia POKE o la orden SYS.

Ejemplos:

```
? USR (90)
100 A=USR (90)
200 IF USR (X)<5 THEN 100
```

Visualiza el valor entregado de la subrutina 90.

3.3. Función SYS (SYSTEM)

SYS puede ser una función programable en lenguaje BASIC o bien una orden.

La función/orden SYS es una función del sistema que transfiere el control del programa a un subsistema independiente.

Formato: | *n SYS (X)* |

n Número de línea.
X Dirección de comienzo.

SYS (abreviatura de SYSTEM) o SYSTEM se utiliza por algunas computadoras para permitir que datos en lenguaje máquina se carguen desde unidades de cinta o de disco en la computadora.

Así pues, se puede utilizar dentro de un programa o desde el exterior.

`SYS 1425`	*Transfiere el control a subrutina de dirección de comienzo 1425.*
`100 SYS (1425)`	*Igual que antes pero en modo programa.*
`200 SYS (A+200)`	*Se bifurca a la subrutina que comienza en la dirección A+200.*

Así, por ejemplo, en las computadoras Commodore el paso de caracteres normales o estándares (caracteres mayúsculas) a caracteres alternativos (letras mayúsculas y minúsculas) se realizan mediante el comando SYS.

`SYS 59468, 14`	*Paso a juego de caracteres alternativo.*
`SYS 59468, 12`	*Vuelta a caracteres estándares.*

3.4. Sentencia CALL

La sentencia CALL transfiere el control del programa a una subrutina en lenguaje máquina, cuya dirección inicial en la memoria viene dada en el formato.

Formato 1:
(IBM PC)

n CALL *varnum* (*variable* 1, *variable* 2,...

n	Número de línea.
varnum	Nombre de una variable numérica que contiene el desplazamiento de la dirección inicial de la subrutina en lenguaje máquina, relativa al argumento actual.
$variable_1$ $variable_2$	Son parámetros opcionales que se pasan como argumentos a la subrutina. Las direcciones de estas variables se introducen en la pila de BASIC cuando se ejecuta la sentencia CALL.

Formato 2:
(Amstrad)

CALL *exprnum [lista de parámetros]*

exprnum expresión numérica.

Formato 3:
(MSX)

n CALL *nombre* (*lista de argumentos*)

4. FUNCIONES ESPECIALES

Dentro de estas funciones especiales podemos considerar diferentes tipos: *Determinación de memoria libre, temporización o reloj, presentación de variables, etc.*

4.1. Funciones MEM-FRE

El cálculo de la memoria libre para el usuario es un dato de interés en la programación, sobre todo si sus programas son largos, de modo que le permita evitar el desbordamiento de la memoria.

4.1.1. FRE

FRE es una función del sistema que proporciona el número de octetos libres disponibles para el usuario.

Formato: | *n* FRE (X) | V = FRE(X) |

n Número de línea.
X Argumento falso que puede ser una expresión o número, se suele utilizar 1 ó 0.

FRE puede aparecer en cualquier parte de un programa, pero generalmente se utiliza en una sentencia PRINT en modo inmediato:

```
PRINT FRE (1)
 16435
```

Si desea saber el espacio de memoria que ocupa un programa ya introducido en su computadora ha de conocer previamente el espacio libre para el usuario nada más encender la computadora. Este dato lo proporciona el fabricante en el Manual y suele aparecer en la pantalla nada más encender la computadora. Así, por ejemplo, la computadora Commodore 64 nada más encenderse aparece el siguiente mensaje:

```
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BASIC BYTES FREE
```

significa que existen 38911 bytes (octetos) libres para el usuario.

Una vez conocido este dato, para conocer la memoria libre para el usuario bastará que realice:

```
PRINT FRE (1)
```

si, por ejemplo, se obtuviera

```
28432
```

el programa que tiene en memoria ocupa 38911−28432=10.489 octetos (bytes). APPLE II, COMMODORE, MSX, Amstrad, IBM PC, etc. poseen esta función.

Caso de que no sepa a priori el número de bytes libres para sus programas realice las siguientes operaciones.

```
OK
NEW              orden de borrado de programas en memoria.
OK
PRINT FRE(0)
 44542           este valor dependerá de su computadora y el tamaño de memoria de su PC.
```

Cargue un programa de una unidad de disco

```
LOAD "Demo 1"
OK
PRINT FRE(0)
 475
```

La diferencia entre los valores devueltos (44542−475) proporcionará el espacio de memoria ocupado por su programa.

NOTA: La función FRE se puede utilizar dentro de un programa para saber en todo momento el espacio disponible y evitar que el programa exceda a la capacidad de memoria del PC. Esto es especialmente importante en máquinas con pequeñas memorias o en programas que creen matrices o «arrays» grandes. Mediante FRE (0) se pueden efectuar limpiezas periódicas para ahorrar gastos inútiles de memoria.

4.1.2. MEM

MEM es una función semejante a FRE, que devuelve el número de octetos de memoria no utilizados. Su empleo es igual que el de FRE.

Computadoras que utilizan esta función son algunas de las que poseen BASIC Microsoft, como TRS-80, DRAGON-64, etc.

Ejemplo:

```
PRINT MEM
 23432
```

4.2. Funciones de reloj

La mayoría de las computadoras suelen tener incorporadas funciones de reloj que permitirán medir el tiempo de ejecución de un programa o bien confeccionar un reloj de tiempo real que pueda utilizar el usuario. Cada computadora suele tener un temporizador o reloj interno que se activa mediante funciones específicas.

Las computadoras de tiempo compartido suelen medir el tiempo desde las doce de la noche hasta la siguiente media noche, mientras que las microcomputadoras autónomas miden el tiempo desde el momento del encendido.

4.2.1. TIMER

Esta función la utiliza las computadoras DRAGON.

Formato: TIMER

Esta función devuelve el valor actual del reloj o temporizador y permite ajustarlo. Entrega un valor comprendido entre 0 y 65535, que son unidades de tiempo igual a 1/50 de segundo.

Si desea poner a cero el reloj utilice:

```
TIMER=0
```

Cuando desee ver el valor que en un instante dado tenga el reloj bastará realizar PRINT TIMER.

Ejemplo 1: Cálculo del tiempo de reloj en segundos

```
100 TIMER=0
110 T=TIMER/50
130 FOR I=1 TO 5000
140 NEXT I
200 PRINT
```

Al ejecutar este programa se pone a cero el reloj en la línea 100, en la línea 120 la variable I se hace igual a TIMER/50, o lo que es igual, T se expresará en segundos cuando se imprima

en la línea 200. Este programa mide el tiempo en segundos que tarda la computadora en ejecutar el bucle vacío I, 5000 veces.

4.2.2. TI-TI$

Las computadoras CBM de Commodore tienen una característica importante que los diferencia de otras marcas en la medida de tiempo, y es su reloj de tiempo real.

El reloj de la computadora CBM conserva el tiempo real en un ciclo de 24, horas en horas, minutos y segundos.

Las variables TI y TI$ (abreviaturas de TIME) representan las dos variables del sistema.

Puesta en hora del reloj

Para poner en hora el reloj utilice la siguiente sentencia de asignación:

```
TI$="HHMMSS"
```

donde

es la hora entre 00 y 23
son los minutos entre 0 y 59
son los segundos entre 0 y 59

El método de puesta en hora del reloj es similar al procedimiento en los relojes digitales de cuarzo. Escriba la hora que desea poner unos pocos segundos antes de la misma o de escuchar la señal horaria por radio o televisión.

```
TI$="080000"
```

Cuando sean las ocho en punto pulse la tecla RETURN y el reloj se pondrá en marcha.

Consulta horaria

Para consultar la hora en cualquier momento o durante un programa bastará teclear en modo inmediato o en modo programa:

```
PRINT TI$
```

y la computadora visualizará la hora expresada en HHMMSS.

La computadora mide el tiempo hasta que se desconecta de la red. Ha de volver a poner en hora el reloj cuando encienda de nuevo la computadora.

Cálculos horarios:

```
10 H=TI                 Cálculo de horas.
20 M=INT (TI/60)        Calcula minutos.
30 S=INT (TI/3600)      Calcula segundos.
```

Medida de tiempo de ejecución de un programa

Realice las siguientes frases:

1. Asignar TI a una variable numérica donde desee que comience la medición del tiempo.
2. Reasignar TI a otra variable al final del período o programa.

3. Restar la primera variable de la segunda e imprimir esta diferencia.

```
10 REM Medida de tiempo
20 A=TI
.
.
.
.
.
500 B=TI
600 PRINT "Tiempo empleado ="; B-A
1000 END
```

4.2.3. TIME$ (*en* TRS-80)

Las computadoras TRS-80 devuelven la hora y la fecha con esta función. Contienen un reloj interno que debe ponerlo en hora para usarlo. Cuando encienda la computadora o se haga un RESET el reloj se pone a cero. Para la puesta en hora deberá ejecutar el siguiente programa:

```
10 DEFINT A-Z
20 DIM TM (5)
30 CL=16924
40 PRINT "Introduzca 6 valores con 2 digitos"
50 PRINT "en el orden siguiente:"
60 PRINT "dia, mes, anualidad, horas, minutos, segundos."
70 INPUT TM(0), TM(1), TM(2), TM(3), TM(4), TM(5)
80 FOR I=0 TO 5
90 POKE CL-I, TM(I)
100 NEXT I
110 PRINT "El reloj esta puesto en hora"
120 END
```

Para saber la fecha y la hora en modo directo o programa bastará hacer:

```
PRINT TIME$
```

4.2.4. *Funciones de reloj y calendario en* IBM PC

El IBM PC posee un reloj interno que puede ser programado por el usuario, mediante funciones exclusiva de tiempo: TIME$ DATE$ y TIMER, y que permiten obtener la fecha y hora del día de trabajo. Estas funciones también han podido ser programadas por el DOS.

DATE$

Carga o devuelve la fecha del día. Puede actuar como sentencia o como variable.

Formato:	*varcad*=DATE$	*Actuación como variable.*
	DATE$=*varcad*	*Actuación como sentencia.*

varcad Variable de cadena válida.

Ejemplos:

```
DATE$-D$                        DATE$="4-10-85"
PRINT DATE$                     DATE$="1-12-83"
```

Observaciones

1. Cuando DATE$ actúa como *variable*, se devuelve la fecha con el siguiente formato:

   mm-dd-aaaa
   mm mes
   dd día del mes
   aaaa año

2. Cuando DATE$ actúa como sentencia, *varcad* es una cadena que se utiliza para cargar o asignarle la fecha actual. Los formatos válidos pueden ser:

   mm-dd-aa
   mm/dd/aa
   mm-dd-aaaa
   mm/dd/aaaa

   el año debe pertenecer al rango entre 1980 y 2099.

Ejemplo de aplicación:

```
10 DATE$="8/7/85"
20 PRINT DATE$

RUN
08-07-1985
```

TIME$

Carga o proporciona la hora actual. Puede actuar como variable o como sentencia.

Formato: | *varcad*=TIME$ | *Actuación como variable.*

| TIME$=*varcad* | *Actuación como sentencia.*

varcad Variable de cadena válida.

Observaciones

1. Cuando TIME$ actúa como *variable*, la hora actual se devuelve como una cadena de 8 caracteres con el formato:

   hh:mm:ss
   hh hora (00 a 23)
   mm minutos (00 a 59)
   ss segundos (00 a 59)

2. Cuando TIME$ actúa como sentencia, *varcad* es una cadena que se utiliza para cargar o asignar la hora actual. Los formatos válidos pueden ser:

hh	Puesta en hora en el rango 0-39; por omisión se elige 00 para minutos y segundos.
hh:mm	Puesta en hora con minutos (rango 0-59); por omisión se elige 00 para segundos.
hh:mm:ss	Puesta en hora con minutos y segundos (rango 0-59).

Ejemplo de aplicación:

```
10 CLS
20 LOCATE 12,20
30 PRINT TIME$
40 GOTO 20
```

Visualización continua de la hora en el centro de la pantalla.

4.3. Funciones especiales

Entre las funciones especiales se podrían citar muchas dado que cada computadora tiene las suyas propias, entre otras, por ejemplo, sentencias y funciones para dibujos y sonidos actualmente populares para trabajos técnicos y juegos.

Por ello nos limitaremos a las más usuales, excepción hecha del tema gráficos y sonidos que dejamos fuera del contenido de este libro.

4.3.1. VARPTR (VAR)

Es la abreviatura de VARIABLE POINTER (Puntero de la variable). Esta función devuelve el valor de la dirección de la posición de una variable. Permite localizar la posición donde están almacenados en memoria el nombre y el valor de esta variable. Si la variable especificada no tiene valor asignado se produce un error.

Formato: VARPTR (X) v=VARPTR (X)

X Nombre de la variable.

Ejemplo:

```
100 A=VARPTR (X)
110 PRINT PEEK (A)
120 PRINT PEEK (A+1)
```

A es la dirección donde está almacenada la variable *X*. La dirección *A* contiene el octeto menos significativo y *A*+1 el octeto más significativo.

Cada variable numérica ocupa su valor dos octetos de memoria.

4.3.2. CLS

Es una orden/sentencia que permite limpiar o borrar la pantalla completamente, llevando a su vez el cursor a la parte superior izquierda.

Es muy útil cuando se desea presentar en pantalla la información de modo más agradable a la vista.

Formato: CLS

Ejemplo:

```
10 CLS
20 PRINT "Demo"
30 PRINT "Comienzo de calculo"
40 CLS
```

A lo largo de un programa se incluye con mucha frecuencia el borrado de pantalla, sobre todo cada vez que se presenta un formato nuevo.

VARIACIONES

La computadora TRS-80 COLOR y el DRAGON-64/200 entre otros utilizan

CLEAR *(n)*

donde n activa el color de fondo de la pantalla, en una escala entre 0 y 8, de acuerdo con la siguiente tabla:

0=NEGRO	3=AZUL	6=AZUL TURQUESA
1=VERDE	4=ROJO	7=MAGENTA
2=AMARILLO	5=BEIGE	8=NARANJA

4.3.3. CLEAR/CLR

La sentencia CLEAR (CLR en Commodore 64 y APPLE II) pone todas las variables numéricas a cero y asigna valores nulos a todas las variables de cadena. Se libera en memoria todo el espacio de las matrices, lo que equivale a apagar la computadora, encenderla y volver a cargar el programa en memoria.

Formato: CLEAR *(o bien* CLR)

```
10 REM Programa demo CLEAR
20 A=500
30 A$="DEMO CLEAR"
40 PRINT "Antes de CLEAR A ="; A
50 PRINT "y A$ ="; A$
60 CLEAR
70 PRINT "Despues de CLEAR A ="; A
80 PRINT "y A$ ="; A$
99 END

RUN

Antes de CLEAR A = 500
y A$ =DEMO CLEAR
Despues de CLEAR A = 0
y A$ =
```

Variante de CLEAR

En BASIC MICROSOFT (TRS-80, DRAGON-64, MSX-BASIC, etc.).

Además del formato ya visto se puede utilizar también el siguiente:

Formato: | *n* CLEAR *m*, *p*. |

n Número de línea.
m Número de octetos que se reservan en memoria para las cadenas.
p Especifica la dirección de memoria superior.

```
10 CLEAR
20 CLEAR 100            Asigna 100 octetos de memoria para cadenas.
30 CLEAR 1000, 32000    Asigna 1000 octetos de memoria y la dirección más alta 32000.
```

4.3.4. *Sentencia* WAIT

Esta sentencia permite detener la ejecución del programa durante un período de tiempo prefijado (pausa) y a su terminación continuará de modo automático el programa en su orden secuencial normal.

Fromato a): | *n* WAIT *m* |

n Número de líneas.
m Número de segundos que se detiene el programa.

Esta sentencia es muy útil para visualizar mensajes detenidamente en pantalla o bien para medir intervalos de tiempo.

Si su computadora no posee esta sentencia puede tratar de sustituirla con bucles vacíos, variando el valor final del índice (T en el ejemplo).

```
100 FOR I=1 TO T
120 NEXT I
```

Las computadoras de HEWLETT PACKARD poseen esta sentencia:

Formato b): | *n* WAIT *np*, *m*, *s* |

n Número de línea.
np Número de puerto (o de dirección específica).
m Máscara.
s Selección.

Se detiene la ejecución de un programa hasta que la entrada del puerto especificado tenga la configuración especificada.

Si s=0 se efectúa una operación AND (Y) entre la entrada del port y la máscara.
Si s≠ se efectúa una operación OR exclusiva entre la entrada del puerto y la selección.

Los datos del puerto se leen repetidamente hasta que el valor sea igual a 1 (o 0 según computadoras), en cuyo caso se reemprende la ejecución del programa.

Computadoras que poseen esta función son: APPLE II, IBM PC, etc.

5. ORDENES/SENTENCIAS DE DEPURACION DE UN PROGRAMA

Como ya hemos indicado en alguna ocasión, será difícil que un programa funcione bien a la «primera». Por ello es preciso disponer de técnicas y herramientas de depuración y corrección de errores a fin de poner a punto el programa.

Las computadoras, y por supuesto la IBM PC no es una excepción, suelen emitir unos mensajes de error cuando el usuario o el programa realizan tareas estándar no previstas por BASIC. Estos mensajes de error (Cap. 11) sirven de ayuda al programador, y normalmente con su lectura y el manual de la computadora suelen ser suficientes para su detección y eliminación.

Sin embargo, algunos errores no están originados por el programa sino que pueden ser el resultado de una entrada inadecuada del usuario (números negativos, en lugar de positivos; divisiones por cero; raíces cuadradas de números negativos, etc.) o algún tipo de problema sencillo relativo a periféricos, como puede ser: impresora apagada, unidad de disco abierta, etc. Evidentemente los mensajes de error, tales como «Illegal function call» (llamada ilegal a función) descorazonan a cualquier programador, máxime si se encuentra en fase de aprendizaje del BASIC. En este párrafo y en el siguiente capítulo le explicaremos técnicas para localización de errores y analizaremos las sentencias específicas de detección de errores durante la ejecución de programas.

Los errores durante la ejecución de programas (*run time error* en inglés), suelen ser difíciles de detectar si no se han previsto durante la fase de programación, ya que pueden ocurrir en diferentes momentos o partes del programa, según sea la respuesta del usuario, periférico, etc. Veamos algunos ejemplos que aclaren estos conceptos.

Ejemplo 1:

```
10 INPUT "Introducir un numero";H
20 PRINT 100/H
30 ...

RUN
Introducir un numero?5
20
Ok

RUN
Introducir un numero?0
Division by zero
Ok
```

El problema reside en que la computadora no puede dividir por cero. El procedimiento para evitar este error, consiste en detectar la entrada 0, y cabrían dos métodos:

a) Advertir previamente al operador con un mensaje idóneo en la línea 10.

```
10 INPUT "Introducir un numero distinto de cero";H
```

b) El procedimiento anterior no evita la parada del programa si a pesar del mensaje el operador se equivoca e introduce 0. Este método sería más idóneo:

```
10 INPUT "Introducir un numero";H
20 IF H<>0 THEN 40
30 PRINT "DISTINTO DE CERO"
40 GOTO 40
..........
```

Ejemplo:

El siguiente programa imprime la raíz cuadrada de un número

```
10 INPUT "Introducir un numero:",N
20 RAIZ=SQR(N)
30 PRINT "La raiz cuadrada de";N;"es igual a:";RAIZ
40 PRINT "Fin"
99 END

RUN
Introducir un numero:64
La raiz cuadrada de 64 es igual a: 8
Fin
Ok

RUN
Introducir un numero:-5
Illegal function call
Ok
```

La computadora no puede extraer la raíz cuadrada de un número negativo. La corrección del programa sería similar al anterior pero en lugar de detectar la entrada 0 , detectar números negativos. Insertar por ejemplo, la línea siguiente:

```
15 IF X<0 THEN PRINT"numeros negativos no permitidos":GOTO 10
```

La detección de errores, sin embargo, suele incluirse dentro de una técnica especial que utiliza sentencias BASIC específicas.

5.1. Captura de errores. Sentencias ON ERROR GOTO y RESUME

La detección o captura de errores (*error trapping* en inglés) permite a un programa continuar su ejecución cuando se produce un error. Aunque su técnica es muy similar a la captura de eventos o sucesos, entrada en el capítulo 9, la detección de errores tiene prioridad sobre la detección de sucesos. En otras palabras, si se produce un error, se desactiva la detección de sucesos de cualquier fuente hasta que la captura del error se ha completado.

La sentencia ON ERROR GOTO se utiliza para bifurcar a una subrutina de tratamiento de errores cuando se produce un error en el programa, sin que se detenga la ejecución del mismo. (En las computadoras Apple II se denomina ONERR GOTO.)

Formato:

n ON ERROR GOTO *m*

n Número de línea.
m Número de línea inicial de la subrutina de captura de errores.

n ON ERROR GOTO 0

Desactiva la captura de errores y los siguientes errores detendrán la ejecución del programa.

La sentencia ON ERROR GOTO debe aparecer en el programa antes de que se pueda encontrar un error. Cualquier error encontrado después de la ejecución de la sentencia ON ERROR GOTO produce una bifurcación al número de línea de comienzo de la subrutina de tratamiento de errores.

La sentencia RESUME cierra la subrutina de tratamiento de errores, y es la última línea de dicha subrutina; es equivalente a la sentencia RETURN.

Formato:

n RESUME [0]	*La ejecución continúa en la línea que provocó el error. (Cero es opcional.)*
n RESUME NEXT	*La ejecución continúa en la línea inmediatamente posterior a la que provocó el error.*
n RESUME *m*	*La ejecución continúa en la línea m.*

Ejemplos:

`10 ON ERROR GOTO 5000`	*Bifurcación a línea 500 si se produce un error.*
`400 ON ERROR GOTO 0`	*Desactivación de la captura de errores.*
`5100 RESUME`	*Salto a línea donde ocurrió el error.*
`5100 RESUME NEXT`	*Salto a la línea siguiente donde ocurrió el error.*
`5100 RESUME 30`	*Salto a línea 30.*

Ejemplo de aplicación:

1. Errores internos al programa.
   El ejemplo mencionado de cálculo de la raíz cuadrada se podría codificar así:

```
10 ON ERROR GOTO 400
20 INPUT "Introducir un numero:",N
30 RAIZ=SQR(N)
40 PRINT "La raiz cuadrada de";N;"es igual a:";RAIZ
50 PRINT "Fin"
99 END
100 '
400 'Subrutina de captura de errores
410 '
420 PRINT "Numeros positivos, siempre:"
430 PRINT "Prueve de nuevo"
440 RESUME 20

RUN
Introducir un numero:-8
Numeros positivos, siempre:
Prueve de nuevo
Introducir un numero: 2
La raiz cuadrada de 2 es igual a: 1.41421
Fin
Ok
```

2. Errores debidos al sistema (periféricos del PC).

```
10 ON ERROR GOTO 400
20 LPRINT "Adelante impresora"
30 PRINT "Impresora preparada"
99 END
100 '
400 'Subrutina de captura de errores
410 PRINT "Impresora no preparada"
420 PRINT "Encienda impresora / Ponga papel"
430 PRINT "Cuando este preparada la impresora"
440 PRINT "Pulse cualquier tecla"
450 T$=INKEY$:IF T$="" THEN 450
460 RESUME

RUN
```

Si la impresora estuviese apagada o no tuviera papel, se visualizarán los siguientes mensajes:

```
Impresora no preparada
Encienda impresora / Ponga papel
Cuando este preparada la impresora      Encender impresora o poner papel.
Pulse cualquier tecla                   Pulsación de una tecla.
Impresora preparada                     Problema resuelto.
```

5.2. Variables ERR y ERL

Las subrutinas anteriores de tratamiento de errores son muy simples y solamente se preveía un tipo de error. Ese no es el caso normal en programas largos. Por ello será preciso normalmente identificar la naturaleza del error, y en el caso de que se produzca más de un error discriminar el tipo de error. Las herramientas que utiliza BASIC para la identificación de errores son las variables (consideradas también funciones) ERR y ERL.

ERR y ERL proporcionan el código de error y el número de línea asociado con un error. Se denominan *variables reservadas.*

ERR devuelve el número del código de error. (ERRN en computadoras HP.)
ERL devuelve el número de línea donde ocurrió el error. (ERRL en computadoras HP.)

Formatos:

IF ERR=*código de error* THEN...

IF ERL=*número de línea* THEN...

IF ERR *código de error* AND ERL < > número de línea THEN...

A las variables ERR y ERL sólo les puede asignar valores BASIC; por eso se denominan *variables reservadas,* lo que significa que no se pueden utilizar esas variables para otros fines que la identificación de errores.

Los códigos de error los proporciona el fabricante en su manual de BASIC. En el caso del IBM PC el Apéndice A del *Manual de Referencia BASIC* enumera los diferentes códigos de los que destacamos los siguientes:

ILLEGAL FUNCTION CALL	- código 5	*(Llamada ilegal a función)*
OUT OF PAPER	- código 27	*(Falta papel)*
DISK NOT READY	- código 71	*(Disco no preparado/puerta abierta)*

El número de línea contenido en la variable ERL cambia cada vez que ocurre un error en una línea diferente; de modo que su aplicación permitirá localizar la línea donde ocurrió el error. Para asegurar que la subrutina de tratamiento de errores se llama, sólo cuando realmente ocurre el error, se puede añadir a la subrutina de captura de errores una nueva línea que compruebe si ha sucedido el error. En el programa de la raíz cuadrada añada la siguiente línea:

```
415 IF ERR<>5 AND ERL<>30 THEN ON ERROR GOTO 0
```

La línea 415 significa que si el código del error no es el número 5 «y» (AND) el error no sucede en la línea 30, si a la línea 0 que tiene el efecto de desactivar la subrutina de captura de errores.

CAPTURA DE DOS O MAS ERRORES

Hasta este momento sólo se ha tratado de la producción de un solo error en un programa; se puede extender la filosofía explicada al tratamiento simultáneo de dos o más errores.

Supongamos que se desea detectar en el programa, ya conocido, de obtención de raíces cuadradas dos errores. Habrá que modificar el programa de la siguiente forma:

```
10 ON ERROR GOTO 400
20 INPUT "Introducir un numero:",N
30 RAIZ=SQR(N)
40 LPRINT "La raiz cuadrada de";N;"es igual a:";RAIZ
99 END
100 '
400 'Subrutina de captura de errores
410 '
420 IF ERR<>5 AND ERR<>27 THEN ON ERROR GOTO 0
430 IF ERR=5 THEN PRINT "Numeros positivos siempre. Prueve
    de nuevo":RESUME 20
440 PRINT "Impresora no preparada"
450 PRINT "Encienda impresora/Ponga papel"
460 PRINT "Cuando este preparada la impresora"
470 PRINT "Pulse cualquier tecla"
480 T$=INKEY$:IF T$="" THEN 480
490 RESUME 40
```

5.3. La sentencia ERROR

ERROR se utiliza en el tratamiento de errores para dos fines: primero, simular la aparición de un error de BASIC (útil para comprobación de sus subrutinas de tratamiento de errores). Segundo, permite la definición de sus propios códigos de error (técnica potente y muy útil para detectar y tratar errores que no son parte del conjunto de errores BASIC, códigos 1a 73).

Formato:

n ERROR *m*

n Número de línea.
m Código del error; rango 1 a 255.

Ejemplo:

```
ERROR 67
too many files
Ok

10 H=25
20 ERROR H #

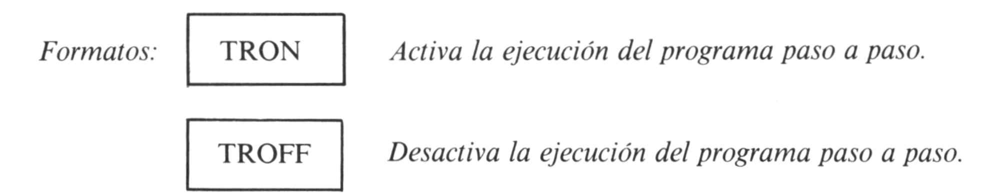
RUN
String too long in line 20
Ok

10 ON ERROR GOTO 500
20 INPUT "Introduzca su numero";N
30 IF N>10000 THEN ERROR 200
      .
      .
      .
500 IF ERR=200 THEN PRINT "NUMERO MAYOR 10000"
510 IF ERL=30 THEN RESUME 20
```

El código de error 200 ha sido definido para el caso de introducción de números mayores que 10000.

5.4. Ordenes TRON y TROFF

Las órdenes TRON y TROFF («trace on» y «trace off» en inglés) se utilizan para *rastrear* el flujo de un programa a medida que se ejecuta. Esta operación en realidad es un seguimiento paso a paso de todas las líneas del programa (la traza del programa).



Formatos: TRON *Activa la ejecución del programa paso a paso.*

TROFF *Desactiva la ejecución del programa paso a paso.*

Cuando se desea ejecutar un programa paso a paso, es preciso activar la orden TRON en modo inmediato. A partir de esta orden se sigue el flujo del programa a medida que se ejecuta visualizándose cada número de línea del programa a medida que se ejecutan, incluidos los números de línea de bifurcación. Para evitar confusión con la salida normal de un programa los números de línea se visualizan dentro de corchetes (por ejemplo [201]).

Las órdenes TRON y TROFF se consiguen pulsando las teclas de función F7 y F8 en el IBM PC. A medida que se rastrea el programa se rellena la pantalla con números y, si desea interrumpir o romper el programa, puede pulsar las teclas CTRL y NUM LOCK. Cuando desee reanudar el programa interrumpido bastará que pulse cualquier otra tecla y los números de línea continuarán visualizándose. Mientras que TRON no se desactive, cada vez que se ejecute un programa, éste se realizará paso a paso.

Ejemplos:

```
TRON
100 FOR I=1 TO 5
110 PRINT I,I^2
120 NEXT I
```

```
RUN
[100] [110] 1               1
[120] [110] 2               4
[120] [110] 3               9
[120] [110] 4               16
[120] [110] 5               25
[120]

120
Ok

10 N=10
20 FOR I=1 TO 2
30 L=N+10
40 PRINT I;N;L
50 N=N+10
60 NEXT
99 END

RUN
[10] [20] [30] [40] 1  10  20
[50] [60] [30] [40] 2  20  30
[50] [60] [99]
Ok
TROFF
Ok
```

A partir de este momento la ejecución de cualquier programa será normal.

VARIANTES DE TRON/TROFF

En lugar de las órdenes TRON y TROFF se suele utilizar en algunas computadoras TRACE y NOTRACE (o bien NOT TRACE).

TRON/TROFF como sentencia

Aunque el método usual de empleo de TRON/TROFF es como orden, se puede utilizar como sentencia para el caso que desee comprobar la ejecución paso a paso de subrutina o partes importantes de un programa.

```
10 . . .
20 . . .
100 TRON
110 . . .
120 . . .
300 TROFF
400 . . .
500 . . .
```

5.5. Sentencia DSP/DISP

DSP es la abreviatura de DISPLAY y se puede utilizar como sentencia u orden, permitiendo visualizar en pantalla una variable específica y sin valor cada vez que a la variable se le asigna un valor; así mismo suele presentar el número de línea de la asignación. Se pueden realizar varias sentencias DSP en un programa, no existiendo un límite fijo.

El empleo como orden para ver el contenido actual de una variable suele ser su mayor aplicación.

Ejemplo:

```
10 DSP A
20 DSP B
```

instruye a la computadora para visualizar (imprimir) las variables A y B y sus valores, así como los números de línea cada vez que se les asigna y reasigna un valor.

La orden RUN desactiva la orden DSP, por lo que para comenzar el programa deberá utilizar GOTO (como orden) o CONT en modalidad inmediato, o bien colocar DSP en líneas de programa.

La desactivación de DSP se realiza también con NO DSP.

HEWLETT PACKARD utiliza en lugar de DSP la palabra clave DISP.

PROGRAMAS RESUELTOS

1. *Aplicación de* ON ERROR GOTO/ERR.

```
10 ON ERROR GOTO 100
20 INPUT "Introducir un numero:",N
30 IF N=0 THEN ERROR 200
40 DIV=1/N
50 PRINT DIV
99 END
100 '
400 'Subrutina de tratamiento de errores
410 IF ERR<>200 THEN ON ERROR GOTO 0
420 PRINT "numeros distintos de cero"
430 PRINT "prueve de nuevo"
450 RESUME 20

RUN
Introducir un numero:0
numeros distintos de cero
prueve de nuevo
Introducir un numero:8
 .125
```

2. *Aplicación de* ON ERROR GOTO/ERR.

```
100 REM Demo "ERR"
110 DIM A(10)
120 CLEAR
130 ON ERROR GOTO 200
140 INPUT "Numero de prueba"; N
150 A(N)=10/N
160 PRINT "Numero"; N; "no produce error"
170 GOTO 140
200 REM Rutina de error
210 IF ERR=9 THEN 230
220 IF ERR=11 THEN 260
225 GOTO 280
230 PRINT N; "Es muy grande"
240 PRINT "N entre 1 y 10"
250 RESUME 120
260 PRINT "El numero menor es 1"
270 RESUME 120
280 PRINT "El numero"; N; "produce un codigo de error"; ERR
300 END

RUN
Numero de prueba? 31
 31 Es muy grande
N entre 1 y 10
Numero de prueba? 0
El numero menor es 1
Numero de prueba? ...
```

3. *Aplicación de* ON-ERROR-GOTO.

```
100 REM Demo "ERL"
110 ON ERROR GOTO 200
120 INPUT "Introducir un numero"; N
130 A=10/(N-1)
140 A=10/(N-2)
150 A=10/(N-3)
160 PRINT "El numero"; N; "no da error"
170 GOTO 120
180 REM Deteccion de error
200 PRINT "El error esta en la linea"; ERL
210 RESUME 120
300 END

RUN
Introducir un numero? 1
El error esta en la linea 130
Introducir un numero? 7
El numero 7 no da error
Introducir un numero? 3
El error esta en la linea 150
..................
```

4. *Aplicación de* ON-ERROR-GOTO.

```
100 REM Demo ON-ERROR-GOTO
110 ON ERROR GOTO 200
120 INPUT "Introducir un numero"; N
130 I=1/N
140 PRINT "El inverso de"; N; "es"; I
150 GOTO 120
200 PRINT "El inverso de 0 no se puede calcular"
210 RESUME 120
400 END

RUN
Introducir un numero? 5
El inverso de 5 es .2
Introducir un numero? 0
El inverso de 0 no se puede calcular
..................
```

5. *Aplicación de* ERR/ERL.

```
10 ON ERROR GOTO 500
20 INPUT "Numero"; X
30 PRINT 10/X
40 GOTO 20
500 REM Rutina de errores
600 PRINT "ERR ="; ERR; "ERL ="; ERL
610 IF ERR=11 AND ERL=30 THEN 700
620 PRINT "Error no conocido"
630 STOP
700 PRINT "X debe ser <> de 0"
800 RESUME 20

RUN
Numero? 0
ERR = 20 ERL = 30
X debe ser <> de 0
```

6. *Aplicación de* SWAP.

```
10 REM Demo "SWAP"
20 INPUT "Introducir dos numeros"; A, B
30 IF A<=B THEN 100
40 SWAP (A, B)
100 PRINT A; "es menor o igual que"; B
110 GOTO 20
200 END

RUN
Introducir 2 numeros? 5, 14
 5 es menor o igual que 14
Introducir 2 numeros? 17, 3
 3 es menor o igual que 17
```

7. *Aplicación de* CLEAR.

```
10 REM Demo "CLEAR"
20 A=1000
30 A$="Hola McKoy"
40 PRINT "Antes de CLEAR A ="; A
50 PRINT "y A$ = "; A$
60 CLEAR
70 PRINT "Despues de CLEAR A ="; A
80 PRINT "y A$ = "; A$
100 END

RUN
Antes de CLEAR A = 1000
y A$ = Hola McKoy
Despues de CLEAR A = 0
y A$ =
```

8. *Aplicación de* SWAP.

```
10 REM Demo "SWAP"
20 A$="Yo que tu" : B$="no lo" : C$="haria"
30 PRINT A$; B$; C$
40 SWAP (A$, C$)
50 PRINT A$; B$; C$
100 END

RUN
Yo que tuno loharia
hariano loYo que tu
```

PROGRAMAS PROPUESTOS

1. Dado el carácter de este capítulo, que puede diferir de un modelo de computadora a otra, le sugerimos que realice los programas propuestos en los Capítulos 7, 8 y 9 incluyendo las siguientes mejoras cuando se considere oportuno.

   a) Sentencias de detección de errores: ON ERROR GOTO/ERR/ERL/RESUME.
   b) Utilizar la sentencia SWAP en los programas de aplicación del Capítulo 9.
   c) Utilización de CLEAR y CLS.
   d) Utilización de TIMER/TI/TIME$ para obtener el tiempo de duración de la ejecución de un programa.

2. Rastrear la ejecución de algunos programas mediante las sentencias TRACE/NOTRACE o bien TRON/TROFF, así como utilizando DSP como orden.

3. Si su computadora posee las sentencias PEEK/POKE, trate de deducir el contenido del bloque de memoria comprendido entre las direcciones decimales 4000 y 5000. Así mismo deduzca cuál es la zona de memoria de pantalla y trate de dibujar figuras mediante estas sentencias en la pantalla, que contengan los símbolos *, # y...

CAPITULO 11

Ficheros/Archivos

0. INTRODUCCION

En este capítulo explicaremos los procedimientos para crear y acceder a colecciones de datos almacenados en discos flexibles, discos rígidos (fijos) y otros dispositivos.

Un *fichero* es una colección de datos o información almacenados bajo un nombre específico y que se guarda en un dispositivo. La información en los ficheros debe ser procesada- recuperada, almacenada o escrita. El tratamiento o procesamiento de ficheros es el objetivo de este capítulo.

A su terminación, el lector habrá aprendido los siguientes conceptos:

— Ficheros.
— Tipos de ficheros: secuencial y aleatorio.
— Estructura de ficheros.
— Creación y lectura de ficheros.
— Modificación y actualización de ficheros.
— Sentencias y órdenes para tratamiento de ficheros.

1. FICHEROS/ARCHIVOS

Como ya se ha comentado, *un fichero* o *archivo** es una colección de información considerada como una unidad. Hay muchos tipos diferentes de ficheros que BASIC utilizará cuando los almacena o conserva en disco y luego los recupera.

Los ficheros pueden contener programas o datos. Los programas se guardan en disquetes con un nombre y pasan a convertirse en ficheros, que serán tratados con una serie de órdenes. Los datos pueden ser cualquier tipo de información que se disponga como una lista o tabla, tal como una agenda de direcciones.

1.1. Ficheros de programas

Los *ficheros programa* típicamente se almacenan en los discos en un formato binario compacto o en un formato ASCII (texto). Estos ficheros creados por las órdenes BASIC, ya conocidas, SAVE, son las que utilizará la computadora cuando se ejecute un programa.

`SAVE "nombre"`	*El programa «nombre» se conserva en formato binario.*
`SAVE "nombre",A`	*El programa «nombre» se conserva en formato ASCII* (tal como aparece en la pantalla).

* El término *archivo* es de uso común en Latinoamérica, mientras que en España se usan ambos términos, con preferencia *fichero.*

La ventaja de la conservación de programas en fichero ASCII es la posibilidad de que sean tratados como ficheros de textos y por consiguiente poder fusionarse o encadenarse varios programas entre sí. La ejecución de programas es independiente del formato en que se hayan conservado.

1.2. Ficheros de datos

Los ficheros de datos manipulan la información que un programa puede procesar o crear. Por ejemplo, un programa de tratamiento de textos será un fichero de datos conteniendo texto. La entrada y salida de datos de programas se consideran ficheros de datos y de este modo se diferencian de los ficheros de programas.

Un fichero, al igual que una matriz o *array,* es un bloque de posiciones de memoria que se pueden utilizar para almacenar información, que debe consistir de datos numéricos o alfanuméricos (cadenas). Sin embargo, hay una diferencia notable entre una matriz y un fichero con respecto al modo en que se almacenan en la memoria de la computadora y a su tratamiento por BASIC. Una matriz almacena información en la memoria interna (memoria de acceso directo, RAM), mientras que un fichero guarda la información en un dispositivo de memoria tal como una cinta magnética, un disco flexible o un disco rígido o fijo.

La sentencia DATA es ideal para almacenar información permanente dentro de un programa (memoria RAM). Sin embargo, hay muchas aplicaciones en que se necesitan introducir o modificar los datos por el usuario del programa, como suelen ser precios de artículos, direcciones, calificaciones de alumnos, expedientes personales, etc. Para conseguir introducir estos datos se requiere gastar memoria interna de la computadora y puesto que ésta suele ser limitada, el medio más adecuado y económico para almacenar gran cantidad de información fuera de la memoria interna de la computadora son los *ficheros de disco.*

El otro procedimiento de introducción de datos en *arrays* es mediante la sentencia INPUT, pero este método tiene el grave inconveniente, para grandes cantidades de datos, que, al ser *temporal,* la información se perderá cada vez que se apague la computadora, con el consiguiente trastorno que supone la nueva entrada de datos. Sin embargo, la información almacenada en discos será siempre permanente y sólo se borrarán los mismos a voluntad del usuario.

1.2.1. Tipos de ficheros

Los tipos de ficheros y su nombre provienen del método en que la información se almacena en un fichero. Los dos ficheros más usuales y que estudiaremos en este libro son: *ficheros secuenciales* y *ficheros aleatorios* o de *acceso directo.*

Las palabras «secuencial» y «aleatorio» especifican el método en que se lee y escribe la información del fichero. Los ficheros secuenciales son como la música grabada en las cintas magnéticas: oir la música del centro de la cinta, supone buscar la melodía deseada. Los ficheros aleatorios o directos, son como los discos de audio: oír la música en una determinada pista exige mover el brazo de la aguja hasta la pista elegida, y a continuación comenzará a sonar la música; un fichero aleatorio es como un disco de un tocadiscos de donde se puede acceder a la información musical de un modo directo o aleatorio mediante el brazo de su tocadiscos.

Ambos tipos de ficheros tienen sus ventajas y sus inconvenientes, su elección depende normalmente del programador, o del tipo de aplicación. Tal vez, para aprender el correcto funcionamiento y la estructura de un fichero, los tipos más adecuados sean los secuenciales y si ya es un programador experimentado, los ficheros directos serán los más idóneos.

Un *fichero,* es un conjunto de *registros* y a su vez, un registro es un conjunto de campos. Un *campo* es el área especificada asignado a un elemento de información («ítem»). Por ejemplo, un fichero de datos de personal de una empresa puede constar de 500 registros (500 empleados) y cada registro contener varios campos de datos sobre el empleado, tales como su nombre, dirección, número de la Seguridad Social, etc. En la figura 13.1 se representa el fichero típico de oficina con sus fichas manuales o tarjetas de papel y de un fichero de computadora en un disquete.

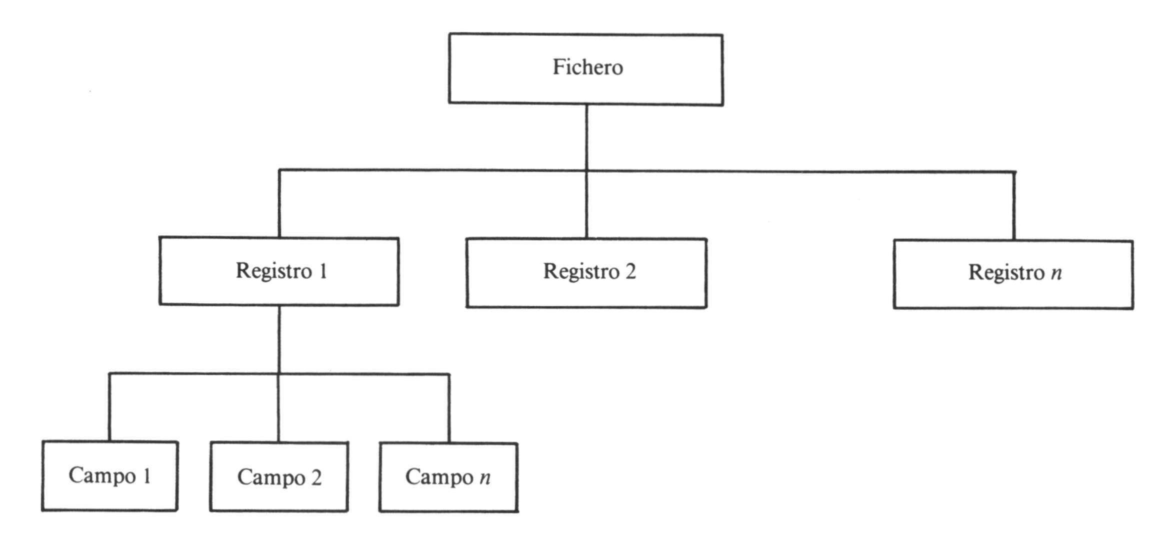


Figura 11.1. Un fichero, un registro, un campo.

En la figura 11.2 se representa la distribución física de un disquete en pistas y sectores (en un disquete para IBM PC se organizan 40 pistas y 9 sectores/pista siendo la capacidad de almacenamiento de cada sector 512 bytes. Por consiguiente en un disquete de doble cara caben 2×40×9×512 bytes o lo que es igual 40×9×½K×2 = 360 K bytes, ya que ½K = 512 bytes).

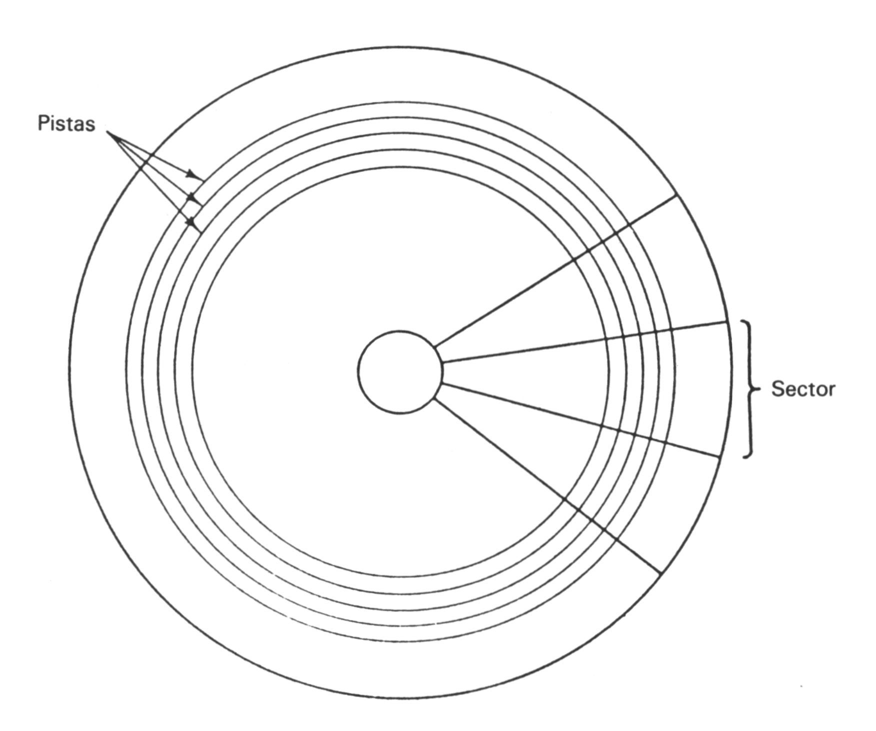


Figura 11.2. Organización física de un disquete (pistas y sectores).

2. ORGANIZACION DE LOS FICHEROS DE DATOS

La organización o estructura de un fichero es el método en que se disponen los registros. La organización de los ficheros afecta a los métodos de identificación y acceso a un registro

particular. Básicamente, existen los dos métodos ya descritos: (1) secuencial, (2) aleatorio o directo. Otras organizaciones de ficheros son: secuencial indexada y secuencial encadenada. La figura 11.3 muestra las cuatro organizaciones, con un ejemplo de 4 registros (1, 2, 3 y 4) en el fichero.

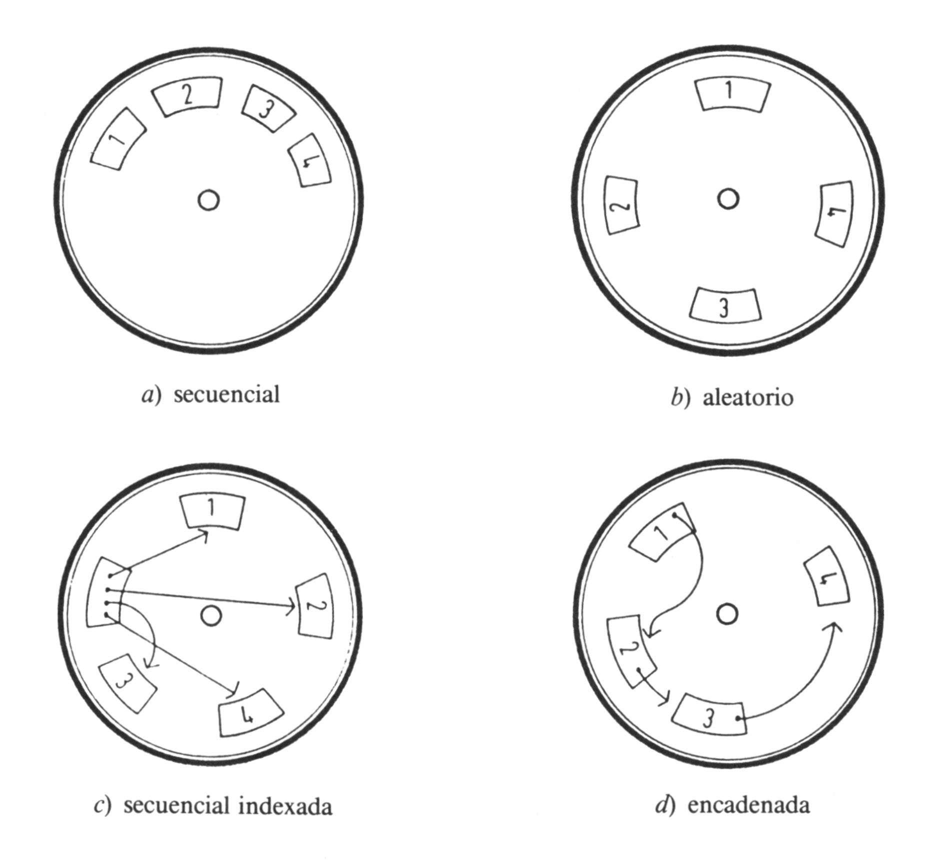


Figura 11.3. Organizaciones típicas de ficheros de datos.

Los registros en los *ficheros secuenciales* se deben escribir, leer o procesar uno después de otro. Esto significa que antes de acceder al registro deseado se deben leer y comprobar todos los registros anteriores. La organización secuencial es común en ficheros cuyos registros se lean y procesen, generalmente, al mismo tiempo. No es necesario que el programador conozca donde están localizados los datos en el disco y sólo conoce este dato la computadora.

La organización de acceso *aleatorio o directo,* en contraste, permite al usuario acceder o cualquier registro sin necesidad de comprobar ningún otro más. Con este método, cada registro debe tener una clave que identifique su posición en memoria. Esta clave puede ser un número, un nombre, un código o simplemente una secuencia de números, conocido normalmente como índice. Tal índice permite al programador y a la computadora localizar cualquier dato siempre que se desee.

Los ficheros secuenciales, tienen registros de longitud variables, mientras que en los ficheros aleatorios, los registros son de longitud fija. Esto significa que cada registro de un fichero secuencial puede tener diferente número de caracteres, mientras que un fichero aleatorio, todos los registros completo cada vez, los ficheros secuenciales, generalmente, requieren menos espacio de memoria que los ficheros aleatorios ya que en los primeros la longitud de cada registro se amolda a sus datos reales y en el segundo será siempre igual con independencia de los datos. Para comprender este concepto, imagínese que uno de los campos de un registro son nombres y apellidos de personas. En los ficheros secuenciales la memoria gastada será la derivada de los nombres y apellidos, mientras que un fichero aleatorio, cada registro tendrá una longitud fija sea cual sea el nombre completo y por ello habrá ocasiones en que se desaproveche memoria al no utilizarse, sobre todo con nombres cortos.

Otra diferencia notable de los ficheros, aunque ya implícitamente se ha expuesto, es que si trabaja con un fichero secuencial y quiere buscar alguna información en un registro, ha de comenzar por el principio del fichero (registro 1) y moverse secuencialmente por el fichero, comprobando en cada uno de ellos los datos, hasta encontrar la información en que está interesado. Si el fichero es de acceso directo, cada registro de su fichero de datos tiene un número de registro y se puede mover directamente a cualquier registro con solo especificar su clase o índice.

Dado que la versión de BASIC más extendida es MBASIC de Microsoft que soporta a las computadoras IBM/PC y compatibles, MSX, Amstrad, Commodore y Apple II bajo CP/M, etc., dedicaremos toda la teoría y ejemplos de ficheros al tratamiento de dichas versiones, incluyendo no obstante al final del capítulo una tabla comparativa de las computadoras comentadas en este libro, las sentencias utilizadas por Apple, HP, Amstrad y Commodore.

3. TRATAMIENTO DE FICHEROS SECUENCIALES

La estructura de un fichero secuencial se muestra en la figura 11.4.

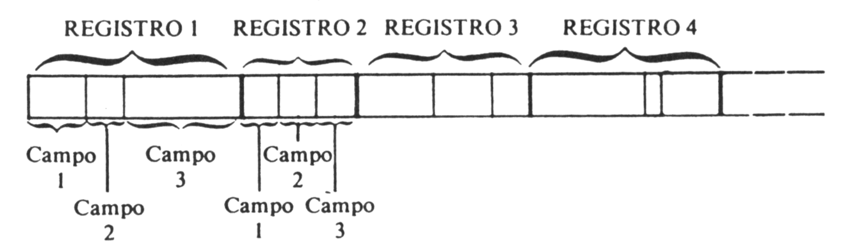


Figura 11.4. Registros de un fichero secuencial.

Un fichero secuencial puede constar de registros de tamaños diferentes. Cuando se lee (extraer información) un fichero secuencial siempre ha de comenzar al principio del fichero; no puede comenzar en un punto aleatorio. Cuando se escribe información en un fichero secuencial, tiene que comenzar o bien al principio o añadirse al final de la información ya almacenada. No se pueden añadir datos en un punto arbitrario del fichero.

La *manipulación* o *procesamiento de un fichero* es el proceso de creación, utilización y mantenimiento de ficheros de datos. El *mantenimiento* o *actualización de un fichero* es la actividad de añadir, modificar o suprimir los datos de un fichero.

El tratamiento de ficheros requiere normalmente estos pasos:

1. Dar nombre a un fichero.
2. Apertura del fichero.
3. Escritura del fichero.
4. Procesamiento-manipulación.
5. Lectura del fichero.
6. Cierre del fichero.

Antes de poder utilizar un fichero (tanto para almacenar como par extraer información de él debe ser «abierto». La operación de *abrir un fichero* (consiste en instruir a la computadora del lugar donde se localiza el fichero en memoria. El primer paso al leer de un fichero con un número de canal desde el cual se leen o escriben datos. Un *canal* es el camino que controla las unidades de entrada-salida. Este proceso se llama «abrir el fichero».

El *cierre del fichero* supone la disociación del fichero después que la lectura o escritura se ha completado. El *procesamiento* puede incluir búsqueda, actualización, cálculo, resumen, análisis o informes. Este paso es común en cualquier proceso, con idependencia de que los datos se almacenen o no en un fichero externo.

Los pasos para acceder a un fichero secuencial son los siguientes: primero, el fichero ha de ser abierto, la *apertura* de un fichero supone la elección del canal y nombre. Una vez que un

fichero ha sido abierto, se puede acceder a él para *salida* «output» (escritura de datos en el fichero) o *entrada* «input» (lectura de datos del fichero). No se puede utilizar un fichero secuencial para salida y entrada al mismo tiempo. En el caso de salida o escritura de datos, se especifica también si los datos se añaden al final del fichero o se añaden al principio. En este último caso, cualquier dato que ya estuviera en el fichero se pierde.

Después que se hayan realizado las operaciones pertinentes con el fichero, debe ser cerrado, siempre se debe cerrar un fichero secuencial abierto, tras la terminación de su manipulación o cuando desee utilizar el fichero en otro modo (salida, entrada o adición de datos). Si *el fichero no se cierra* adecuadamente, los datos pueden perderse.

3.1. Nombres de los ficheros

Un fichero se identifica por un nombre. Los nombres de un fichero BASIC, tanto secuenciales como de acceso directo, se componen del nombre propiamente dicho y de una extensión cuyas características son:

- El nombre del fichero puede ser uno a ocho caracteres de longitud.
- La extensión puede ser uno a tres caracteres de longitud.
- Un nombre de fichero puede incluir los siguientes caracteres:

  A Z
  0-9
  # & $ @ ! % ′ ( ) –

Ejemplos válidos de nombres de fichero son:

```
DEMO1.BAS
NOMBRE#
NOTAS%
```

3.2. Apertura de un fichero. Sentencia OPEN

Los ficheros se *crean* (escritura de datos en el fichero) y se leen por programas BASIC. Tanto para la operación de creación o de escritura como para la lectura, el fichero debe ser abierto previamente, sin cuyo requisito es imposible acceder al fichero.

La sentencia OPEN crea un canal a un fichero residente en un dispositivo específico.

Formato:

1. *n* OPEN *espefich* FOR *modo* AS [#] *numfich*

2. *n* OPEN *modo2, [#]* *numfich, espefich*

1. *n* — Número de línea.
   espefich — Especificación de fichero (dispositivo: nombre · extensión) dispositivo = unidad de disco.
   modo — OUTPUT - salida (escritura)
   INPUT - entrada (lectura)
   APPEND - añadir registros.
   numfich — Número de fichero, 1 a 3.

2. *modo2* O - salida (escritura)
I - entrada (lectura)
R - entrada/salida al azar.
numfich Número de fichero, 1 a 3.
espefich Especificación de fichero.

Precaución:

- Si un fichero secuencial existente se abre para salida (escritura de datos), los datos originales se destruyen.
- Cuando se añaden datos a un fichero, éstos se situarán al final del fichero existente.
- La longitud de registro sólo suele utilizarse en ficheros de acceso directo.

Ejemplos:

```
10 OPEN "NOMBRES" FOR INPUT AS #1
50 OPEN "TELEFONOS" FOR OUTPUT AS #2
60 OPEN "VENTAS.MAC" FOR OUTPUT AS #1
100 OPEN "A:DEMO1" FOR OUTPUT AS #1
200 OPEN "I",3,"A:TEST.RAN"
```

3.3. Escritura en un fichero. Sentencias PRINT#, WRITE#

Una vez que un fichero ha sido abierto para escritura o salida («output»), se puede escribir información en el fichero especificado, mediante la sentencia PRINT# y PRINT# USING.

Formato:

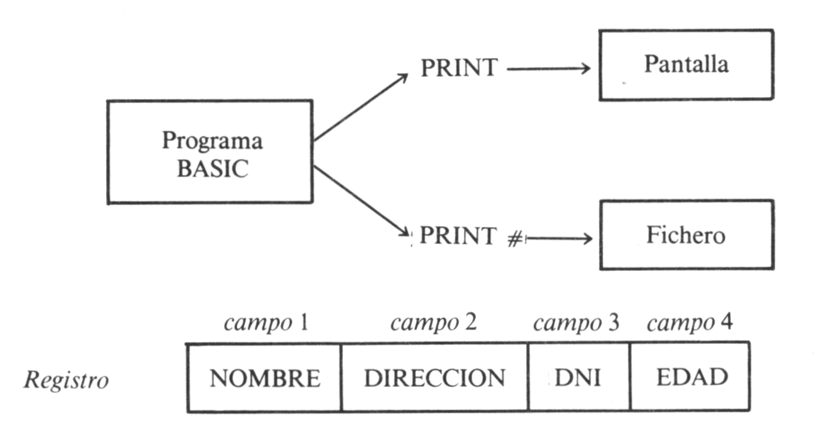
n PRINT# *numfich, expresiones*

n Número de línea.
numfich Número del fichero utilizado en la apertura.
expresión Expresiones numéricas o alfanuméricas.

n PRINT# *numfich,* USING a$; *expresiones*

a$ *Expresión de cadena del formato de impresión.*

Las sentencias *PRINT#* funcionan igual que PRINT, excepto que éstas escriben el registro en la pantalla.
El esquema siguiente ilustra la relación entre PRINT y PRINT#.




```
100 PRINT#1, NOMBRE$;DIRECCION$;DNI$;EDAD
```

La línea anterior escribirá en el fichero de disco, un registro que consta de cuatro campos: NOMBRE, DIRECCION, DNI y EDAD. PRINT# graba en el fichero una imagen de los datos tal como se visualizan en pantalla, por ello deberá utilizar delmutadores adecuados.

En un registro se pueden escribir múltiples datos, pero deben estar separados adecuadamente.

Datos numéricos:

```
100 PRINT#1,A      imprime el valor de A en el fichero 1
110 PRINT#1,B      imprime el valor de B en el fichero 1
120 PRINT#1,C      imprime el valor de C en el fichero 1
```

Al final de cada sentencia PRINT se realiza una marca «fin de datos» y en consecuencia se guardarán 3 valores independientemente que se pueden grabar en disco. Sin embargo,

```
100 PRINT #1,A,B,C
```

se escriben los tres valores A, B y C como una sola cadena de dígitos. Por consiguiente, si se desea escribir tres valores diferentes A,B,C, se debe utilizar un separador (tal como una coma) entre cada dos elementos.

```
100 PRINT#1,A;B;C
```

Datos alfanuméricos (cadenas):

```
10 A$="DNI"
20 B$="757467"

110 PRINT#1,A$;B$
```

Se graba o conserva en el fichero DNI757467. Si desea dejar un espacio en blanco entre ambas cadenas, deberá utilizar un delimitador explícito.

```
100 PRINT#1,A$;",";B$
```

En esta ocasión se graba DNI,757467

```
100 PRINT#1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

se graba «DNI»«757567»

Sentencia WRITE:

WRITE# y PRINT# son muy similares y las dos escriben datos en un fichero. Se diferencian en que WRITE inserta comas entre los elementos según se graba y delimita las cadenas con comillas

```
A$="DNI":B$="757467"
WRITE#1,A$,B$
```

graba en el fichero la información

```
"DNI","757467"
```

Estas dos cadenas podrán ser recuperadas con los formatos anteriores con otras dos variables de cadena.

3.4 Cierre de un fichero. Sentencia CLOSE

Después que se ha procesado un fichero es necesario «cerrarlo». La sentencia CLOSE cierra el fichero.

Formato:

n CLOSE # *numfich1*, # *numfich2*, ...

n	Número de línea.
numfich	Número de fichero.
variable	Almacena los datos del fichero.

```
70 CLOSE#1            Cierre del fichero 1
90 CLOSE#1,#4         Cierre de los ficheros 1 y 4
100 CLOSE             Cierre de todos los ficheros
```

Las sentencias END, NEW, RESET y SYSTEM cierran también todos los ficheros que estén abiertos.

3.5. Lectura de un fichero. Sentencias INPUT # y LINE INPUT

La información almacenada en un fichero puede ser recuperada por examen, modificación y borrado utilizando una sentencia PRINT. La sentencia INPUT se refiere a la entrada de información del disco en lugar del teclado.

El orden en que se leen los datos ha de ser idéntico al orden en que fueron escritos en el fichero, es decir, el tipo de datos que se leen y asigna a una variable debe concordar con el tipo de variable, de lo contrario resultará un error «Type Mismatch».

Cuando INPUT busca datos, todos los espacios no significativos, retornos de carro de línea se ignoran (son delimitadores).

Ejemplos:

```
100 INPUT#1,PRECIOS
```

lee un número almacenado en el fichero abierto con el número 1 y asigna este número a la variable PRECIOS

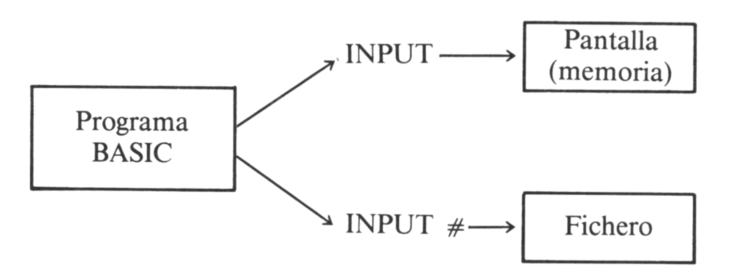
```
100 INPUT#1,PRECIOS,CANTIDAD
```

lee un registro y asigna el primer campo a la variable PRECIOS y el segundo campo a la variable CANTIDAD

```
200 INPUT#1,ALIMENTO$;CLASES;RACION$;PRECIO
```

lee el registro de cuatro campos y asigna cada campo a las variables ALIMENTO$, CLASES, RACION$ y PRECIO.

El esquema siguiente ilustra la relación entre la sentencia INPUT e INPUT #.



Sentencia LINE INPUT #:

Las cadenas que incluyan comas y puntos y comas no pueden ser leídas por la sentencia INPUT#, ya que ésta las interpreta con delimitadores y separadores de cadena.

Por ejemplo, la sentencia

```
100 PRINT#1,"Mackoy,Jose Luis"
```

escribe en el fichero

```
Mackoy,Jose Luis
```

Cuando INPUT lee este dato, interpreta a Mackoy y José Luis como dos cadenas independientes.

La sentencia INPUT# lee de un fichero de igual forma que LINE INPUT lee del teclado, ambas leen e interpretan una línea completa incluyendo comas y puntos y comas en una sola cadena.

Formato: | *n* LINE INPUT# *numfich, varcadena* |

n Número de línea.
numfich Número de fichero.
varcadena Variable de cadena.

Ejemplos:

```
100 LINE INPUT#1,NOMBRE$
```

Si en el ejemplo inicial del párrafo, hubiese leído el registro «Mackoy, José Luis», si se probara con una sentencia PRINT se tendrá:

```
PRINT NOMBRE$
Mackoy,Jose Luis
```

3.6. La marca fin de fichero (EOF)

Cuando se braban o conservan los datos en un fichero, BASIC graba automáticamente al final del mismo, una marca denominada *fin de fichero* (EOF «end of file»).

Al leer un fichero secuencial, si se intenta leer después de determinar el fichero, o lo que es igual leer más registros de los que existen solamente, (por ejemplo, tratar de leer 101 registros, cuando el fichero sólo tiene 100 registros), se obtiene un mensaje de error

`INPUT PAST END IN N` *n* Número de línea.

que significa «lectura más allá del final».

La función EOF indica si el final físico de un fichero ha sido alcanzado.

Formato: | *n* v=EOF *(numfich)* |

n Número de línea.
v Variable.
numfich Número del fichero que se esté leyendo.

Si el último elemento que se ha leído es el último del fichero, EOF es verdad (−1); si no EOF es falsa (O). Un medio para utilizar esta función es como parte de una sentencia IF-THEN.

Ejemplos:

```
100 IF EOF(1) THEN 300
           ↑
           └── número de fichero

500 IF EOF(4) THEN PRINT "Final del fichero no.4"
```

Captura de errores

Puesto que ocurre un error cuando se intenta leer pasado el final de fichero, se puede utilizar la sentencia ERROR (ver capítulo 10) para detectar el final de un fichero. Situando una sentencia ON ERROR GOTO XXX, antes de que se utilice una sentencia INPUT #, se transfiere el control a la línea XXX si ocurre un error. El programa puede comprobar el valor del código del error para asegurarse que el error cometido fue precisamente «pasar el final del fichero».

Por consiguiente, se debe detectar la condición *fin de fichero* ants que ocurra un error. Es decir, la función EOF o la sentencia ERROR deben aparecer en el programa antes que se ejecute una sentencia INPUT # o afín.

3.7. Ampliación de un fichero secuencial

Con frecuencia será necesario añadir registros (información) a un fichero secuencial que ya existe. Por ejemplo, si desea actualizar un inventario de repuestos de un almacén, necesitará periódicamente añadir los nuevos repuestos. A primera vista parece simple el problema: utilizar la sentencia OPEN para abrir el fichero y FOR OUTPUT para añadir datos. Sin embargo, esas operaciones son incorrectas, ya que como se ha comentado en párrafos anteriores, tan pronto se *abre* un fichero para *salida* se destruye su contenido actual. La solución es utilizar una variante del formato de la sentencia OPEN diseñada específicamente para añadir datos a un fichero existente.

OPEN *espefich* FOR APPEND AS# *numfich*

Ejemplo:

```
100 OPEN "DEMO13." FOR APPEND AS#2
```

Siempre que los datos se escriban con PRINT#, se añadirá al final del fichero existente.

OPEN FOR APPEND se puede utilizar también para crear un fichero que no exista anteriormente.

4. MANIPULACION DE UN FICHERO SECUENCIAL

Una vez analizado en el párrafo anterior todo el estudio relativo a ficheros secuenciales, vamos a afrontar la creación de un fichero secuencial (escritura en un disquete o disco fijo) con un ejemplo práctico de todo el proceso.

Para crear un fichero secuencial, necesita seguir estos pasos:

1. Abrir un fichero para salida (escritura).
2. Aceptar datos desde el teclado (o de otro medio).
3. Escribir los datos en el fichero.
4. Cerrar el fichero.

El siguiente ejemplo muestra los pasos anteriores.
Las líneas 10-90 son la introducción al programa propiamente dicho y preparación de la unidad de disco.

```
10 '***FICHERO SECUENCIAL***
20 '
30 'Este programa crea un fichero secuencial
40 'en la unidad A.
50 CLS:KEY OFF          'Borrado de pantalla y teclas de funcion
60 PRINT "Coloque su disco para ficheros en la unidad A"
70 PRINT "Pulse cualquier tecla cuando este preparado"
80 IF INKEY$="" THEN 80
90 CLS
```

4.1. Creación del fichero

Se trata de crear un fichero que contenga 30 números elegidos por el usuario en introducidos por teclado:

```
100 '---creacion de fichero---
110 DIM A(30)     'dimensionado matriz de numeros
120 '---apertura---
130 OPEN "A:NUMEROS" FOR OUTPUT AS#1
140 '---aceptacion de datos desde el teclado---
150 FOR I=1 TO 30
160 INPUT "Numero"; N(I)
170 NEXT I
180 '---Grabacion de datos en disco---
190 FOR I=1 TO 30
200 PRINT #1,N(I)
300 NEXT I
310 CLOSE 1     'Cierre del fichero 1
400 END
```

Los dos bucles I se pueden agrupar en uno sólo del siguiente modo:

```
150 FOR I=1 TO 30
160 INPUT N
170 PRINT#1,N
180 NEXT
```

De este modo, desaparecen las líneas 150 a 300 del programa.

4.2. Lectura del fichero

Una vez que el fichero está creado y guardado en el disco, se puede recuperar o acceder al fichero cuando se desee; para ello, se han de seguir los siguientes pasos:

1. Abrir el fichero como entrada (lectura).
2. Lectura de datos del fichero.
3. Manipular y procesar los datos.
4. Cerrar el fichero.

El siguiente ejemplo muestra los pasos anteriores:

```
500 '---lectura fichero secuencial---
510 '
520 '---apertura---
530 OPEN "A:NUMEROS" FOR INPUT AS#1
540 FOR I=1 TO 30
550 IF EOF(1) THEN 590            'Detencion EOF
560 INPUT#1,N(I)           'Lectura del fichero
570 PRINT N(I)             'Proceso de datos
580 NEXT
590 CLOSE#1                'Cierre del fichero
999 END
```

Recurriendo a un bucle WHILE- WEND, se pueden sustituir las líneas 540 a 580 del bucle FOR-NEXT, por las siguientes:

```
540 WHILE NOT EOF(1)
550 INPUT#1,N(I)
560 PRINT N(I)
570 WEND
```

Este bucle significa que, mientras no se alcance el final del fichero 1, se continue la ejecución del bucle entre las sentencias WHILE y WEND. Este método es más adecuado por la lectura de datos que el típico bucle FOR-NEXT.

4.3. Actualización de un fichero

La actualización de un fichero secuencial exige su lectura total previa del fichero y una nueva grabación de datos. Dos procedimientos se pueden seguir según que la lectura y escritura se realice de forma conjunta o independiente.

Procedimiento 1:

Los pasos a seguir son:

a) *Abrir* el fichero para entrada y leer el fichero completo en una o más matrices (una por cada campo). *Cerrar* el fichero.
b) Efectuar los *cambios* en el elemento correspondiente.
c) Abrir el fichero para *salida* (se borra el fichero original) y escribir las matrices en el fichero original. *Cerrar* el fichero.

Para ver una aplicación práctica, crearemos primero un fichero, por ejemplo, de nombres de alumnos.

```
100 '---Escritura del fichero---
110 '
120 OPEN "Nombres" FOR OUTPUT AS#2
130 'LINE INPUT permite introducir comas en el nombre
140 FOR I=1 TO 50
150 LINE INPUT "Introducir nombre ",NOMBRE$
160 PRINT#2,NOMBRE$
170 NEXT I
180 CLOSE
199 END
```

El programa de modificación del fichero «Nombres» es el siguiente:

```
400 '---Cambios en el fichero "nombres"---
410 '
420 DIM NOMBRE$(50)
430 OPEN "Nombres" FOR INPUT AS#2
440 FOR I=1 TO 50
450 INPUT#2,NOMBRE$
460 NOMBRE$(I)=NOMBRE$
470 NEXT
480 CLOSE
490 '
500 'Modificacion
510 INPUT "numero de alumno a modificar",ALUMNO
520 INPUT "Nuevo nombre?",NUEVONOMBRE$
530 '
540 '
550 OPEN "Nombres" FOR OUTPUT AS#2
560 FOR I=1 TO 50
570 IF I=ALUMNO THEN NOMBRE$(I)=NUEVONOMBRE$
580 WRITE#2,NOMBRE$(I)
590 NEXT
600 CLOSE
999 END
```

La línea 570 cambia el nuevo nombre por el antiguo y a continuación se escribe en el fichero.
Habrá observado el lector que el fichero ha sido leído completo en primer lugar, luego se ha borrado en la línea 550 (FOR OUTPUT) y por último se ha vuelto a escribir con el mismo nombre pero ya con las modificaciones realizadas.

Procedimiento 2:

Los cambios realizados anteriormente incluyen: lectura de los registros del fichero, modificaciones y escritura de los registros en un nuevo fichero. El proceso se esquematiza en la figura 11.5.

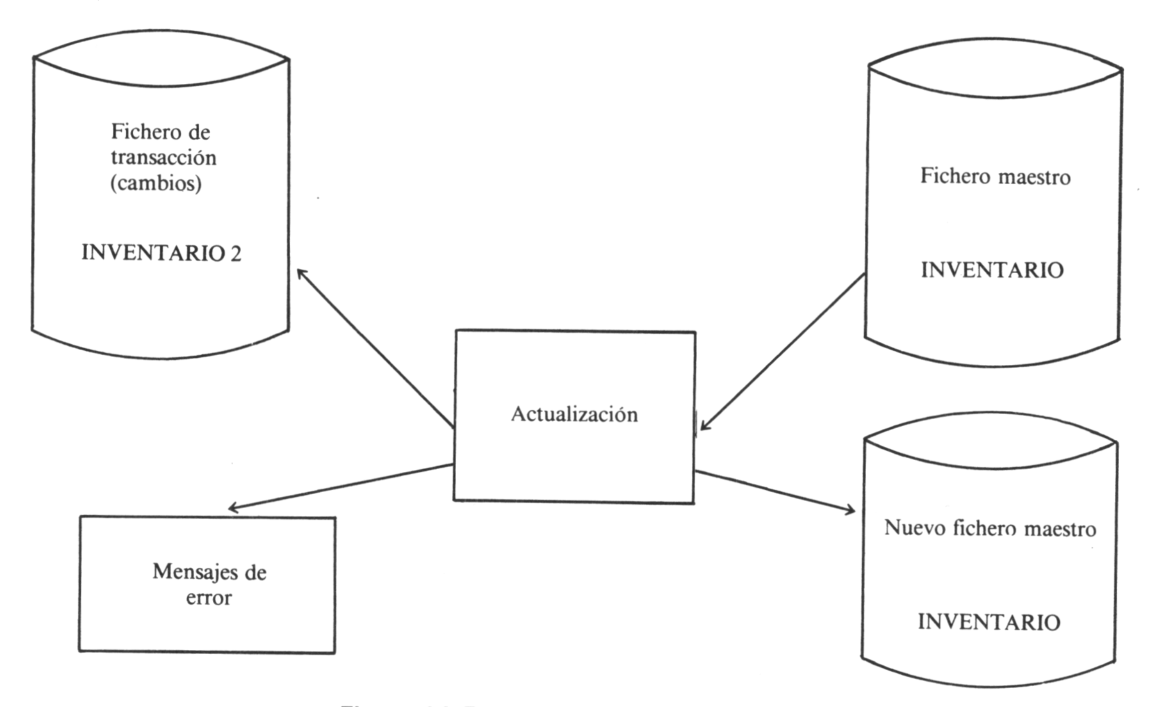
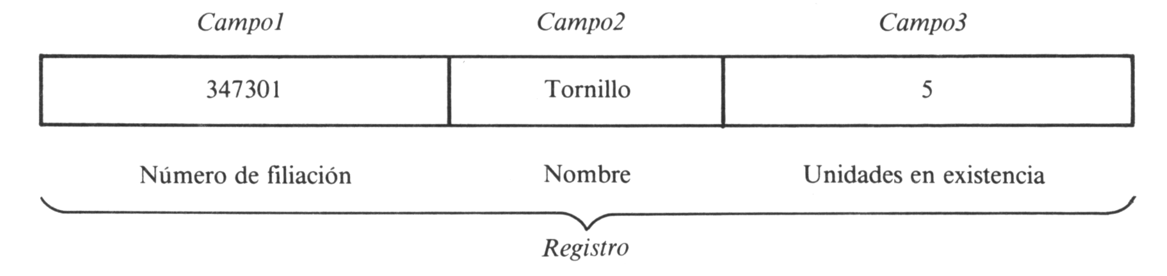


Figura 11.5. Actualización de un fichero.

En realidad se elige un fichero de auxiliar o intermedio que solo interviene para realizar la transacción (equivale en realidad a los cambios de matrices del procedimiento 1).

Supongamos un fichero de inventario que posea 100 registros y cada registro con tres campos (número de filiación del elemento, nombre del elemento, y número de unidades disponibles):



El siguiente programa actualiza el fichero INVENTARIO (grabado con el número 1), con modificaciones del número de unidades existentes, después de cada operación de venta. El proceso a seguir es:

a) Abrir el fichero INVENTARIO como # 1 para entrada.
b) Abrir el fichero INVENTARIO 2 como #2 para salida (fichero auxiliar).
c) Repetir el proceso hasta el final del fichero:

  c.1 Lectura de un registro del fichero 1; al final saltar al paso *d*.
  c.2 Hacer cambios.
  c.3 Escribir el registro en el fichero 2.

d) Cerrar ambos ficheros.
e) Borrar el fichero antiguo INVENTARIO, con la orden KILL.
f) Cambiar el nombre INVENTARIO 2 por INVENTARIO, con la orden NAME.

El programa resultante es:

```
10 'Actualizacion fichero INVENTARIO
20 OPEN "I",#1, "INVENTARIO"
30 OPEN "O",#2, "INVENTARIO 02"
40 FOR I=1 TO 100
50       IF EOF(1) THEN 140
60       INPUT#1,FI,NOMBRE$,UNIDADES
70       PRINT "incremento/disminucion del elemento no."FI,NOMBRE$
80       PRINT "Escriba la reduccion con signo negativo."
90       INPUT CAMBIO
100      UNIDADES=UNIDADES+CAMBIO
110      PRINT#2,FI;",","NOMBRE$;",";UNIDADES
120 NEXT I
130 PRINT "Terminada operacion"
140 CLOSE#1,#2
999 END
```

En este momento, en el disco existen dos ficheros: INVENTARIO con datos sin actualizar; INVENTARIO 2 con datos actualizados. El proceso a seguir ahora serían los pasos *e* y *f*.

`KILL "INVENTARIO"` *borra el fichero INVENTARIO*

`NAME "INVENTARIO2" AS "INVENTARIO"`

cambio de nombre del fichero «INVENTARIO2» por INVENTARIO

5. FICHEROS DE ACCESO DIRECTO

La característica fundamental de un fichero directo o aleatorio es su organización o estructura que permite leer o escribir un registro dado, directamente sin necesidad de lecturas o escrituras intermedias. Si se desea acceder a un fichero específico, basta escribir una instrucción específica en BASIC. La rapidez es otro de los factores importantes que hace a esta técnica particularmente interesante para la gestión de datos.

Un fichero se organiza en *registros* y cada registro contiene uno o más campos.

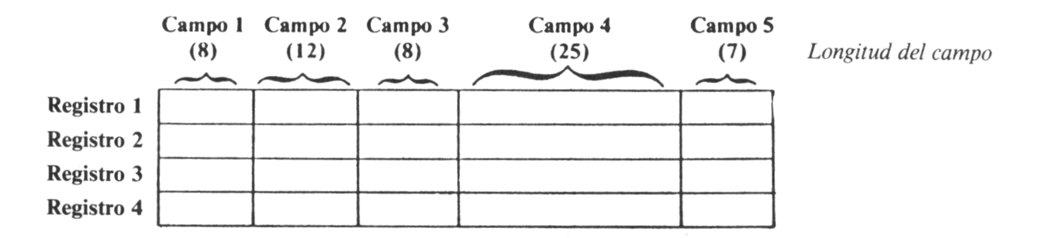
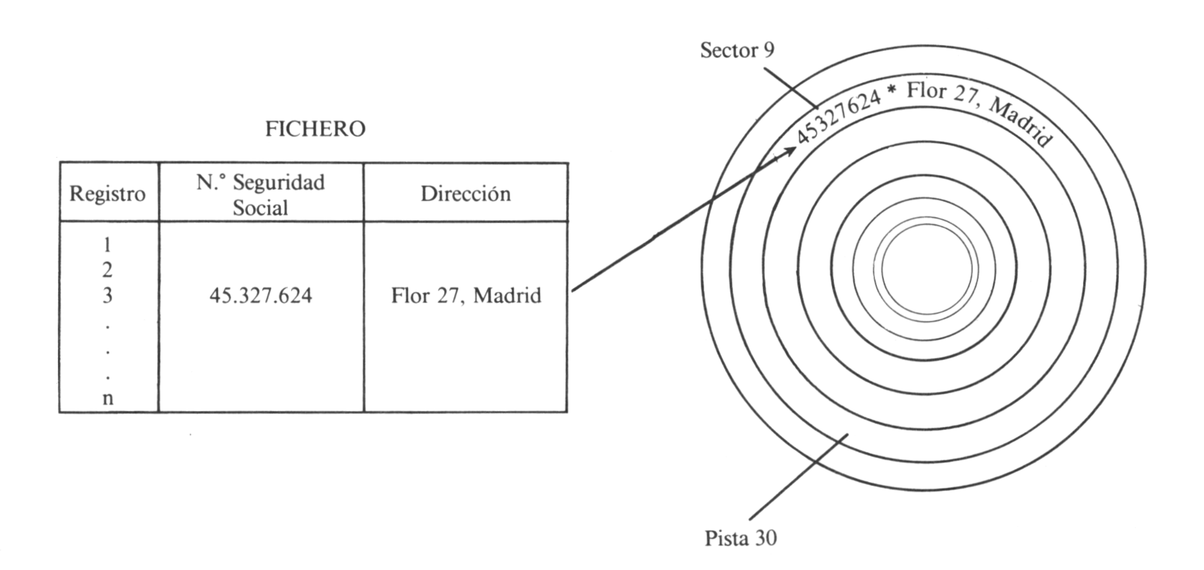


Figura 11.6. Registros de un fichero de acceso directo.

Los registros de un fichero directo están numerados y se accede a ellos por este número o clave. Una *clave* es el campo que identifica a un registro. Por ejemplo, el número de la Seguridad Social, el DNI, etc.

Los ficheros directos se almacenan en pistas de discos o disquetes. Cualquier sector del disco se puede identificar por su posición en la primera pista (ver figura 11.6).


FICHERO

Registro	N.° Seguridad Social	Dirección
1		
2		
3	45.327.624	Flor 27, Madrid
.		
.		
.		
n		

Figura 11.7. Organización física de ficheros en disquetes.

5.1 Acceso a ficheros directos de disco

Cuando una computadora lee o escribe datos en/desde un fichero, los datos realmente van a una *memoria intermedia* «buffer». Un *«buffer»* es una zona de almacenamiento temporal de la memoria que actúa como enlace entre un programa y un fichero de disco y donde los datos esperan su proceso.

El proceso de acceder a un fichero directo se muestra en la figura 11.8.

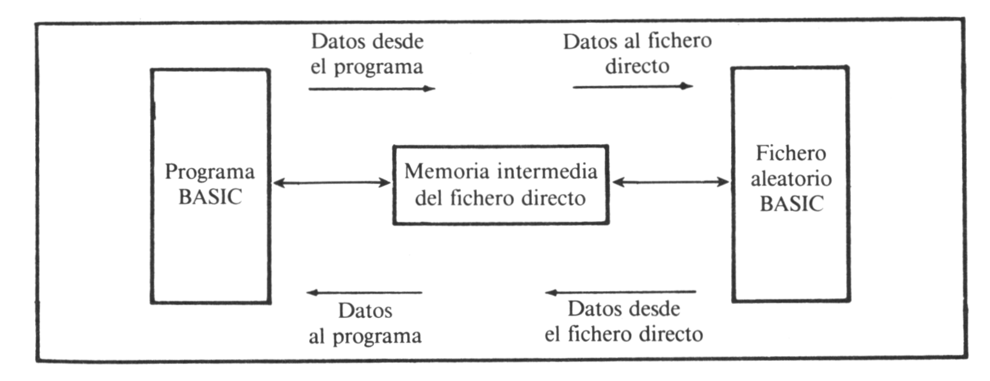


Figura 11.8. Proceso de acceso a un fichero.

Para crear un registro en un fichero, el programador necesita crear el registro, transferirlo a la memoria intermedia y a continuación escribirlo en el fichero.

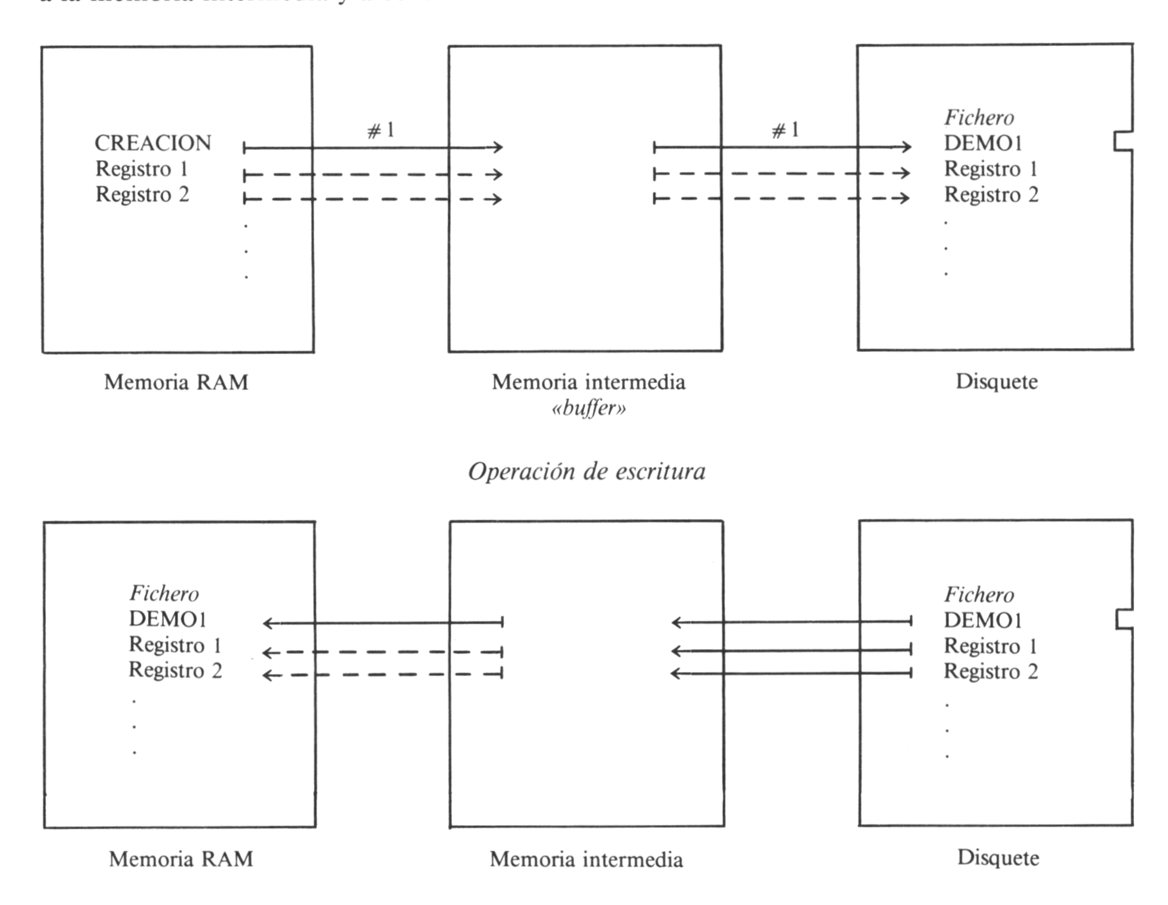


Figura 11.9. Lectura/escritura en disco.

Por consiguiente, el tratamiento o gestión de ficheros directos requiere los pasos previos de definición de los registros y campos, la transferencia de datos de la memoria intermedia al disco o viceversa.

Los pasos generales son:

1. Abrir el fichero como un fichero de acceso directo.
2. Definir un registro definiendo los campos y su longitud en el registro.
3. Trasvasar los datos desde el disco a la memoria intermedia o viceversa.

4. Procesar los datos.
5. Cerrar el fichero.

Como ya conoce el lector, el orden de las anteriores tareas dependerá de la operación a realizar con el fichero.

5.2. Apertura y cierre de ficheros (OPEN/CLOSE)

Los formatos de las sentencias OPEN y CLOSE son similares a las ya vistas en los ficheros secuenciales.

OPEN

Formatos:

1. *n* OPEN *espefich* AS [#] *numfich* [LEN=*longr*]

2. *n* OPEN"R", [#] *numfich, espefich [,longr]*

Longr Expresión entera que fija la longitud del fichero en el rango 1 a 32767; si se omite este parámetro la longitud del registro es 128 bytes.

R indica entrada/salida al azar.

CLOSE

Formato: *CLOSE# numfich*

Ejemplos:

```
OPEN "R",#1, "DIRECCIONES"
OPEN "R",#1, "DIRECCIONES",40
CLOSE    #4
```

5.3. Definición del formato de un registro. Sentencia FIELD

Cuando se utilizan ficheros secuenciales, el programador no tiene que preocuparse de la longitud de los registros. El *registro lógico* (conjunto de datos escritos con PRINT) no tiene longitud fija, y el PC le asigna su valor exacto, con un *registro físico* de 512 bytes (rector de disco).

En los ficheros de acceso directo, el programador ha de definir la longitud o tamaño del registro. La longitud física del registro sigue siendo 512 bytes, sin embargo, la longitud lógica del registro está determinada por el programador y puede ser inferior a 512 bytes. En los ficheros secuenciales se aprovecha mejor el espacio que en los ficheros directos. En éstos, si un cierto campo, por ejemplo, un nombre lo ha definido con una longitud de 25 bytes (25 caracteres) ésta será la longitud máxima de cualquier nombre que se introduzca en ese campo, pero si los nombres son más cortos de 25 bytes, ocuparán también 25 bytes, lo que supone mayor ocupación de espacio de memoria.

La sentencia FIELD es una sentencia ejecutable, que define el registro, sus campos y sus longitudes.

Formato:

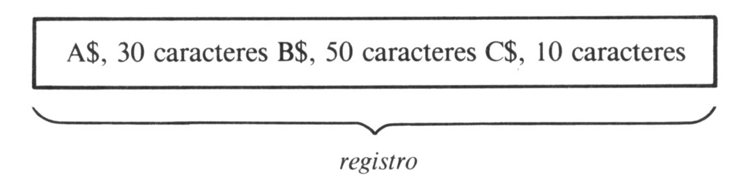
n FIELD [#] *numfich, longitud 1* AS *varcad1,* AS [longitud2 As varcad2]...

numfich Número del fichero en sentencia OPEN.
anchura Longitud de cada campo (en bytes).
varcard1 Nombre del campo1 del registro.

La suma de todas las *longitudes* de los campos no puede ser mayor que longr (longitud del registro) definido en la sentencia OPEN; en caso contrario, se producirá un error «Field Overflow» (Desbordamiento de campo).

Ejemplos:

```
10 FIELD 1, 30 AS A$, 50 AS B$, 10 AS C$
```



El registro contiene 3 campos de 30, 50 y 10 bytes de longitud.

```
10 FIELD#2, 25 AS NOMBRE$, 40 AS DIRECCION$
```



5.4. Escritura de datos en la memoria intermedia

Las sentencias LSET y RSET sitúan una cadena en un campo de la memoria intermedia.

Formato:

n LSET $varcar_1$ = $varcad_2$ *(o una expresión)*

n RSET $varcard_2$ = $varcad_2$ *(o una expresión)*

n Número de línea.
$varcard_1$ Variable de cadena.

La variable de cadena $varcad_1$ ha tenido que definirse en la sentencia FIELD como nombre de un campo de la memoria intermedio del fichero directo.

Cuando se utiliza LSET, los espacios en blanco, si existen se sitúan a la derecha del campo; es decir, la cadena se justifica a derecha. Si se utiliza RSET, los espacios en blanco se sitúan a la izquierda del campo; es decir, las cadenas se justifican a la derecha.

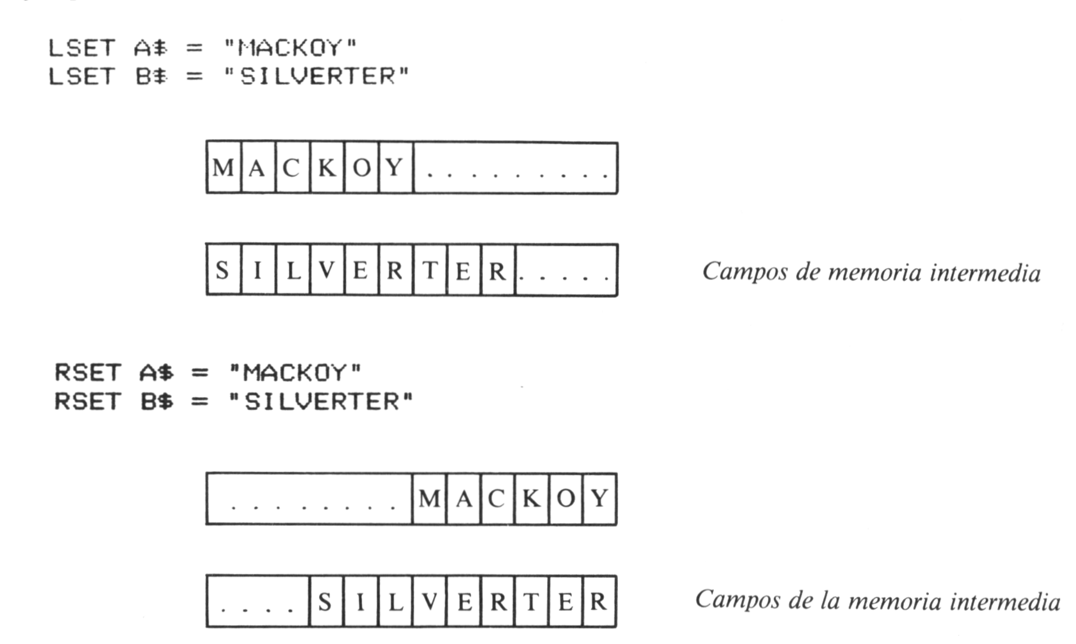
Por ejemplo, si A$ = «Juan» y B$ = «García».

```
100 LSET N$ = A$ + " " + B$
```

situará «Juan García» en el campo N$ de la memoria intermedia.

Si una variable de campo se origina, sin LSET o RSET, no puede utilizarse para colocar datos en un campo de un registro.

Ejemplo:



Como en las sentencias FIELD solo pueden aparecer variables de cadenas, significa que los ficheros directos sólo almacenan variables de cadena y si ha de trabajar con números deberá convertir previamente a cadena cualquier número que desee almacenar. Las *funciones* MKS$, MKD$ y MKI$ realizan estas tareas.

MKI$

Convierte un entero en una cadena de dos bytes.

V$ = MKI$ *(expresión entera)*

MKS$

Convierte un número de simple precisión en una cadena de 4 bytes.

V$ = MKS$ *(expresión de simple precisión)*

MKI$

Convierte un número de doble precisión en una cadena de ocho bytes.

V$ = MKD$ *(expresión de doble precisión)*

Una vez que se hayan convertido los valores numéricos a cadenas, se utiliza LSET y RSET pero sólo con las variables representativas de los campos.

Ejemplo 1:

Supongamos que se desean cuatro números en un registro. Dos de los números son enteros, uno es de simple precisión y otro de doble precisión. La sentencia FIELD que define el registro es la siguiente:

```
100 FIELD#1,2 AS A$, 4 AS B$, 2 AS C$, 8 AS D$
```

Después de generar los valores que se han de almacenar, las siguientes sentencias almacenarán las varibles numéricas correspondientes, Q, en la memoria intermedia del fichero.

```
400 LSET A$ = MKI$ (M%)
410 LSET B$ = MKS$ (N)
420 LSET C$ = MKI$ (P%)
430 LSET D$ = MKD$ (Q)
```

Estas sentencias llenarán la memoria intermedia de la forma siguiente:

A$	B$	C$	D$
M%	N	P%	Q

Ejemplo 2:

```
100 FIELD#1,2 AS XI$, 4 AS XS$, 8 AS XD$
110 X$ = MKI$ (X)
120 LSET XI$ = X$
130 LSET XS$ = MKS$ (4.85)
140 LSET XD$ = MKD$ (4.347865853123)
```

La línea 100 define una memoria intermedia que contendrá tres cadenas de caracteres de 2, 4 y 8 caracteres. La línea 110 convierte el valor X en una cadena de 2 bytes.

Las líneas 120, 130 y 140 convierten la variable.

La línea 120 sitúa la cadena X$ en la zona XI$ de la memoria intermedia. La línea 130 convierte el valor 485 en una cadena de 4 bytes y lo sitúa en la zona XS$ de la memoria intermedia. La línea 140 convierte el valor 4.347592653589798 en una cadena de bytes.

Cuando se lee un registro de un fichero secuencial (con la sentencia GET), que contiene valores numéricos, han de convertirse de nuevo a valores numéricos antes de que pueda procesarlos en su programa. Las funciones se utilizan para convertir las cadenas a su valor inicial. Son:

CVI, CVS y CVD

Tabla 11.1 Funciones de conversiones número/cadenas.

Número	*Conversión números/cadena*	*Conversión cadenas/números*
Entero	A$ = MKI$ (M%)	M% = CVI (A$)
Simple precisión	B$ = MKS$ (N)	N = CVS (B$)
Doble precisión	C$ = MKD$ (P#)	P# = CVD (C$)

```
100 FIELD#1, 2AS A$, 4 AS B$, 8 AS C$
110 GET#1, R
120 A% = CVI (A$)
130 B = CVS (B$)
140 C# = CVD (C$)
150 PRINT A%, B, C#
```

La memoria intermedia contiene tres cadenas de caracteres A$, B$ y C$. La línea 110 lee un registro. Las líneas 120 a 140 convierten las cadenas en valores numéricos.

5.5. Escritura/lectura de ficheros

Cuando se han rellenado todos los campos de un registro mediante sentencias LSET o RSET, se debe utilizar PUT para escribir el registro en el fichero.

Formato: *n* PUT [#] *numfich, [numreg]* [ ] *opcional #*

n Número de línea.
numfich Número de fichero.
numreg Número del registro; es opcional y si se omite, se escribe el registro contenido en la memoria intermedia.

Ejemplos:

```
10 PUT #1,4
```
Escribe el registro 4 del fichero 1.

```
10 PUT #A, B-1
```
Escribe el registro B - 1 del fichero A.

La lectura de un fichero de acceso directo es muy similar a su escritura. GET permite trasferir la información de un registro del disco a la memoria.

Formato: *n* GET # *numfich, [numreg]*

n, numfich y *numreg* Igual que en PUT

Ejemplo:

```
10 GET #1,2
```
Copia el contenido del registro 2 en la memoria intermedia.

```
100 GET 2,I
```
Copia el contenido del registro I del fichero 2 en la memoria intermedia.

Lectura de datos de la memoria intermedia

Una vez que la sentencia GET ha cargado un registro en la memoria intermedia del fichero de acceso directo; se puede acceder a los datos, o bien utilizando las variables de cadena definidas en la sentencia FIELD o bien tratando la memoria intermedia como un fichero secuencial e introduciendo caracteres con una sentencia INPUT = o LINE INPUT #.

6. GESTION DE FICHEROS DE ACCESO DIRECTO

Las etapas a seguir para crear un fichero de acceso directo son más complejas que las requeridas para crear un fichero secuencial. El programador es responsable de la apertura de caracteres, del rellenado de la memoria intermedia y por último de la escritura del registro. El orden de estas diferentes etapas es importante y debe ser registrado.

La secuencia en que se deben ejecutar dichas etapas para la *creación de un fichero* es:

1. Apertura de un fichero existente o un nuevo fichero.

   OPEN "R", *n, "nombre del fichero",* LE

2. Declaración de la memoria intermedia correspondiente a ese fichero.

   FIELD # *n, long* AS *var* $long_2$, AS var_2, ...

3. Rellenando de la memoria intermedia.

   LSET o RSET

4. Escritura del registro en la posición deseada.

   PUT # n, *numreg*

5. Cierre del fichero cuando se han escrito todos los ficheros.

   CLOSE *n*

6.1. Aplicación práctica de un fichero de acceso directo

Se desea diseñar y conservar en un disco un fichero de acceso directo que contenga los datos fundamentales de las ventas realizadas en unos grandes almacenes. Estos datos se deben almacenar en registros que deben tener 5 campos:

- Código del artículo.
- Nombre del vendedor.
- Unidades vendidas del artículo.
- Precio del artículo.
- Fecha de la venta (formato *dd/mm/aa*).

Diseño del fichero

Se debe especificar:

— Nombre del fichero.
— Longitud y campos del fichero.
— Número de registros.

Nombre del fichero de datos: VENTAS. MAC

Registro: 5 campos
CODIGO, NOMBRE, ARTICULO, PRECIO, FECHA

Nombre del campo	*Tipo*	*Anchura* (caracteres)	*Descripción*
CODIGO$	Cadena	5	Código del artículo (5 dígitos)
NOMBRE$	Cadena	30	Nombre y apellidos del vendedor
ARTICULO%	Número	2	Unidades vendidas (entero)
PRECIO	Número	4	Precio (número de simple precisión)
FECHA$	Cadena	8	Fecha de la venta (formato DD/MM/AA)

Longitud del registro

49 (caracteres o bytes) (suma anchura o longitud de los 5 campos).

El número de registros no es preciso definirlo previamente; bien un bucle o un contador pueden contar el número de ellos. Las operaciones de escritura y lectura del fichero las diseñaremos como subrutinas independientes.

Apertura del fichero

```
30 OPEN "R",#1, "VENTAS.MAC", 49
```

nombre del fichero
longitud de un registro

Si se omitiera el número 49 para la longitud del registro, se supondría una longitud de registro de 128 caracteres o bytes, lo que supondría un gasto de espacio de disco innecesario.

Definición del registro

```
40 FIELD#1,5 AS R1$, 30 AS R2$, 2 AS R3$, 4 AS R4$, 8 AS R5$
```

Los cinco campos son R1$, R2$, R3$, R4$ y R5$.

Subrutina de escritura de un registro

```
1000 REM***escritura registro REG%***
1010 LSET R1$ = CODIGO$
1020 RSET R2$ = NOMBRE$
1030 RSET R3$ = MKI$ (ARTICULOS%)
1040 RSET R4$ = MKS$ (PRECIO)
1050 LSET R5$ = FECHA$
1060 PUT #1, REG%
1070 RETURN
```

Subrutina de lectura de un registro

```
2000 REM***Lectura del registro numero REG%***
2010 GET #1, REG%
2020 CODIGO$ = R1$
2030 NOMBRE$ = R2$
2040 ARTICULOS% = CVI (R3$)
2050 PRECIO = CVS (R4$)
2060 FECHA$ = R5$
2070 RETURN
```

Entrada de datos

```
100 REM ... entrada de datos ...
110 CLS
120 PRINT "Numero de registro"; REG%
130 PRINT "Introducir los valores de los campos"
140 PRINT: PRINT: PRINT
150 INPUT"Codigo? (0 = Fin)"; CODIGO$
160 IF CODIGO$ = "0" THEN 240 ' "0" Bandera/marca fin de datos
170 INPUT "Nombre"; NOMBRE$
180 INPUT "Cantidades vendidas"; ARTICULOS%
190 INPUT "Precio"; PRECIO
200 INPUT "Fecha (DD/MM/AA)"; FECHA$
210 GOSUB 1000 'llamada a subrutina escritura registro
220 REG% = REG% + 1 'Numero de registro
230 GOTO 100 'Nueva entrada de datos
240 CLOSE 1      ' # es opcional y aqui no se pone
```

Lectura del fichero

```
300 REM Lectura del fichero
310 CLS:PRINT "MENU"
320 PRINT: PRINT: PRINT "<1> Fichero completo"
330 PRINT "<2> Un registro"
340 INPUT "Seleccione opcion"; OPCION
350 IF OPCION = 1 THEN 500
360 IF OPCION = 2 THEN 600
370 IF OPCION >=3 AND OPCION <1 THEN 310
380 '
390 '
500 FOR REG% = 1 TO 1000
510     GOSUB 2000
520     IF LEFT$ (CODIGO$,1) = "0" THEN 540
530 NEXT REG%
540 '
550 '
600 INPUT "Numero de registro";REG%
610 GOSUB 2000
620 CLOSE 1
```

Evidentemente, una vez que haya leído el fichero completo o un registro, se puede tratar o proceder a voluntad del usuario. Esta tarea no se ha realizado y se deja en la mano del lector, si así lo desea.

7. FUNCIONES LOC/LOF

Las funciones LOC y LOF trabajan con ficheros secuenciales y ficheros de acceso directo.

LOC

1. *Ficheros secuenciales.* Devuelve el número de registros leídos desde o grabados en un fichero desde que se abrió.
2. *Ficheros aleatorios.* Devuelve el número de registro del último registro leído o grabado en un fichero de acceso directo o aleatorio.

Formato: LOC *(numfoch)*

numfich Número de fichero utilizado el abrir dicho fichero.

LOF

Devuelve el número de bytes asignados al fichero (longitud del fichero).

Formato: | LOF *(numfich)* |

NOTA:

Estas funciones ignoran la longitud del registro especificado en la sentencia OPEN y suponen que todo el fichero está compuesto de registros de 128 bytes. Por consiguiente el valor devuelto por LOF es siempre un múltiplo de 128. Si el número de bytes en el fichero número 1 está entre 1 y 128, el valor de LOF(1) es 128. Si el número de bytes está entre 129 y 256, el valor de LOF(1) es 256, y así sucesivamente.

De igual modo, si el último registro leído o escrito en el fichero 1, está contenido en los primeros 128 bytes del fichero, el valor LOC(1) es 1. Si está en los segundos 128 bytes del fichero, entonces el valor LOC(1) es y así sucesivamente.

Tratamiento de ficheros diferentes en microcomputadoras

COMPUTADORAS APPLE II

Nombre de ficheros: Hasta 30 caracteres.

PREFIJO

D$ equivalente a CTRL/D o bien CHR$(4)
D$=CHR$(4)
D$ alerta al sistema de que lo que sigue es una orden de fichero en lugar de un mensaje o valor a visualizar.

FICHEROS SECUENCIALES

Sentencia OPEN (apertura de un fichero)

*n*PRINT D$; «OPEN <*nombre de ficheros*>»
100 D$=CHR$(4)
110 PRINT D$; «OPEN DEM01»

Sentencia CLOSE (cierre de un fichero)

n PRINT D$; «CLOSE <*nombre de fichero*>»
90 D$=CHR$(4)
100 PRINT D$; «OPEN NOMBRE»
110 PRINT D$; «CLOSE NOMBRE»

Sentencia WRITE (almacenar datos en un fichero)

n PRINT D$; «WRITE <*nombre de fichero*>»

Sentencia DELETE (borra un fichero de texto)

n PRINT D$; «DELETE <*nombre de fichero*>»

Sentencia READ (recuperar información de un fichero)

n PRINT D$; «READ <*nombre de fichero*>»

Sentencia APPEND (añadir datos a un fichero)

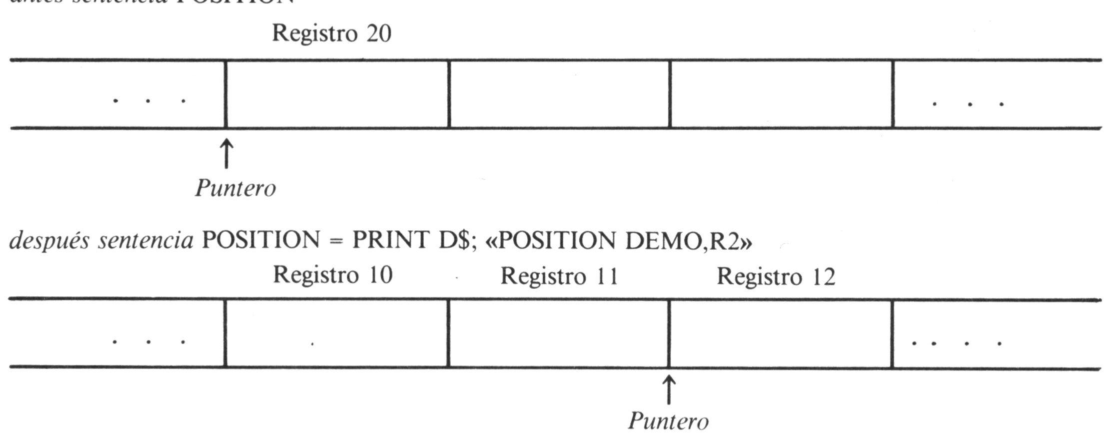
n PRINT D$; «APPEND <*nombre de fichero*>»

Sentencia POSITION (moverse a un registro específico)

La sentencia POSITION permite mover el puntero desde su posición actual en el fichero al principio de cualquier registro dentro del fichero. Cualquier dato que se escriba posteriormente comenzará en la nueva posición.

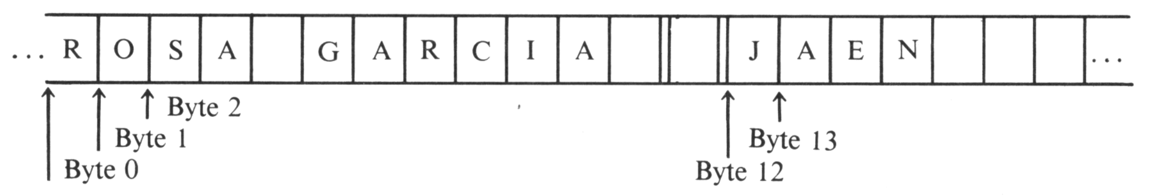
n PRINT D$; «POSITION <*nombre de fichero*>, Rm»
m número de registros a saltar desde la posición actual a la nueva posición en el fichero.

antes sentencia POSITION



EL PARAMETRO BYTE (B)

Se puede especificar el número de byte (carácter) con las sentencias READ y WRITE para mover el puntero del fichero un número especificado de carácteres.



n PRINT D$; «READ <*nombre de fichero*>, Bm»

o bien,

n PRINT D$; «WRITE <*nombre de fichero*>, Bn»
n PRINT D$; «WRITE POEMA, B13»

FICHEROS ALEATORIOS

Sentencia OPEN (apertura de fichero)

n PRINT D$; «OPEN <*nombre de fichero*>, *Lm*»
m número de caracteres del registro
10 PRINT D$; «OPEN DEMO, L100»

el fichero DEMO tiene registros de 99 caracteres y un byte para el retorno de carro.

Sentencia WRITE (escritura de datos en un fichero)

n PRINT D$; «WRITE <*nombre de fichero*>, *Rm*»
100 PRINT D$; «WRITE DEMO2, R50»
R50, número de registro 50

Sentencia READ (lectura de datos en un fichero)

n PRINT D$; «READ <*nombre de fichero*>, *Rm*»
100 INPUT «Registro deseado: »;N
110 PRINT D$; «READ »; F$;» ,R»;N
220 INPUT B$

EL PARAMETRO BYTE

Indica una posición comenzando en el principio del registro.

n PRINT D$; «WRITE <*nombre de fichero*>, *Rn, Bm*»

o bien,

n PRINT D$; «READ <*nombre de fichero*>, *Rn, Bm*»

BASIC ANSI (HP-85, HP-86, HP-87, BASIC HP)

Nombre de ficheros: Longitud máxima, 6 caracteres.

FICHEROS SECUENCIALES

Apertura para escritura o lectura: Sentencia ASSIGN #

n ASSING # *m* TO *«nombrefichero»*

m canal o memoria intermedia de salida

Escritura de datos: Sentencia PRINT #

n PRINT # *m* ; variables

Cierre de fichero: ASSIGN

n ASSIGN # *m* TO *

Lectura de datos: READ

n READ # *m;* variables

FICHEROS DE ACCESO DIRECTO

Creación: CREATE

n CREATE *«nombre del fichero», m, l*

m número de registros *l* longitud del registro

Apertura: ASSIGN

n ASSIGN # *m* TO *«nombre del fichero»*

Escritura: PRINT #

n PRINT # *m,* I; *lista de variables*

I número del registro

Cierre del fichero

n ASSIGN # *m* TO *

Lectura: READ

n READ # *m*,I; *lista de varibles*

Tabla 11-2. Sentencias y órdenes de tratamiento de ficheros.

BASICA GW BASIC (IBM PC)	MBASIC (Amstrad con CP/M)	HP 85 86 87	APPLE II	COMMODORE (Sólo opción casete)	MSX	ACCION *Ordenes (Comandos)*
FILES	FILES	CAT	CAT	• LOAD "$",8 y LIST • CATALOG	DIR	Catálogo o directorio de los ficheros disponibles en un disco, casete o cartucho.
FORMAT	FORMAT	ERASETAPE	Opción en menú principal	INITIALIZE	FORMAT	Inicialización *(formateado)* del disco.
NAME	NAME	RENAME	RENAME	RENAME	NAME	Cambio de nombre del fichero.
KILL	KILL	PURGE	DELETE	SCRATCH	KILL	Destrucción o borrado de un fichero.
SAVE	SAVE	STORE	SAVE	SAVE	SAVE	Almacenaje de un programa en un disco, casete o cartucho.
LOAD	LOAD	LOAD	LOAD	LOAD	LOAD	Carga de un programa desde la memoria externa a memoria central.
RUN *nombre*	RUN *nombre*	—	—	—	—	Carga y ejecución del programa.
MERGE	MERGE	—	—	—	MERGE	Fusión de programas de número de línea distinto.
CHAIN	MBASIC	CHAIN	CHAIN	—	—	Carga y encadenamiento de programa.
—	—	CREATE	CREATE	—	—	Creación de un fichero de datos.
						Ficheros secuenciales
OPEN"O",*n,espefich*	OPEN"O",*n,espefich*	ASSIGN	OPEN *nombre*	OPEN *nf,nd,ds, nombre*	OPEN *nombre* FOR *modo* AS *numfich*	Apertura de un fichero de datos en creación (modo "O") o en acceso (modo "I").
OPEN"I",*n, espefich*	OPEN"I",*n, espefich*					
OPEN *espefich* FOR *modo* AS *numfich*	OPEN *espefich* FOR *modo* AS *numfich*					
CLOSE # *num*	CLOSE # *num*	ASSIGN TO*	CLOSE *nombre*	CLOSE # *num*	CLOSE # *num*	Cierre del fichero.
PRINT # *num,datos*	PRINT # *num,datos*	PRINT#,*datos*	WRITE *nombre*	PRINT#, *datos*	PRINT#, *datos*	Registro o grabación de datos en memoria externa.
WRITE #	WRITE #	—	—	—	—	—
EOF *numfich*	EOF *numfich*	—	EOF *numfich*	—	EOF *numfich*	Marca fin de fichero.
LINE INPUT #	LINE INPUT #	—	—	—	LINE INPUT #	Lectura de datos de memoria externa.
LOF *numfich*	LOF *numfich*	—	—	—	LOF *numfich*	
INPUT #*num,var*	INPUT #*num,var*	READ#,	READ *nombre*	INPUT #*num,var*	INPUT #*num,var*	Lectura de datos de memoria externa.
						Ficheros acceso directo
OPEN "R",[#]*numfich, espefich,[,longr]*	OPEN "R",[#]*numfich, espefich,[,longr]*	—	OPEN *nom* Ln1,Sn2, Dn2,Vn4	—	—	Apertura de un fichero.
OPEN *espefich* AS[#] *numfich*/LEN=*longr*	OPEN *espefich* AS[#] *numfich*/LEN=*longr*	—	—	—	—	—
FIELD #*num*,l1 AS var1, ...	FIELD #*num*,l1 AS var1, ...	—	—	—	FIELD #*num*, l, AS *var*1, ...	Definición de la asignación de espacio para variables.
LSET/RSET	LSET/RSET	—	—	—	LSET/RSET	Puesta de datos de la memoria central a la memoria intermedia.
PUT # n_1,*num*	PUT # n_1,*num*	—	WRITE *nombre*,R1	—	PUT # n_1,*num*	Los datos del *buffer* se transfieren al disquete (escritura).
GET # n_1,*num*	GET # n_1,*num*	—	READ *nombre*,R1	—	GET # n_1,*num*	Los datos del disquete se transfieren al buffer *(lectura)*.
CVI/CVS/CVD	CVI/CVS/CVD	—	—	—	CVI/CVS/CVD	Conversión de variables de cadena en variables numéricas.
MKI$/MKS$/MKD$	MKI$/MKS$/MKD$	--	—	—	MKI$/MKS$/ MKD$	Conversión de variables numéricas en variables de cadena.

espefich = unidas de disco: nombre, extensión; *num* = número de canal; *var* = variable; n_1 = número de registro; *numfich* = número de fichero.

PROGRAMAS RESUELTOS

NOTA PREVIA:

Recuerde que las sentencias INPUT tienen dos formatos; en los siguientes programas utilizamos ambos.

1. *Creación de un fichero secuencial denominado «CIUDADES» en el disco de la unidad A. La terminación del fichero se realizará a voluntad del usuario cuando introduzca la marca final de datos del fichero«*».*

```
10 CLS: KEY OFF 'Borrado pantalla y teclas de funcion
20 PRINT "Entrada disquete en unidad A"
30 PRINT "Pulse cualquier tecla, cuando este preparado"
40 IF INKEY$ = "" THEN 40
50 CLS
60 OPEN "A:CIUDADES" FOR OUTPUT AS #1
70 PRINT "Introducir nombre ciudad"
80 INPUT "Para terminar pulse *", C$
90 IF C$ = "*" THEN 140
100 PRINT #1, C$
110 GOTO 70
120 '
130 '
140 CLOSE
```

2. *El programa siguiente acepta los datos climatológicos del día (temperatura, humedad y precisión atmosférica) introducidos por teclado y los conserva o graba en un fichero denominado «TIEMPO». Los registros tienen tres campos, T,H,P.*

```
100 OPEN "O",#1, "TIEMPO"
110 PRINT "Escriba la temperatura, humedad y presion"
120 INPUT T,H,P
130 PRINT#1,T;",";H;",";P
140 IF T = -999 THEN 160      '-999 marca fin de datos
150 GOTO 120
160 CLOSE #1
999 END
```

3. *Lectura de un fichero que contiene el nombre de cliente y el número de unidades de un producto adquiridas diariamente. Los registros solo tienen un campo N$.* El programa visualiza el fichero y la media de unidades adquiridas.

```
10 OPEN "I",#1, "CLIENTES"
20 PRINT "NOMBRE CLIENTE", "NO. DE UNIDADES"
30 IF EOF (1) THEN 110
40 LINE INPUT#1, N$
50 N=N+1
60 ORDEN$ = MID$ (N$, 21, 3)
```

```
70 UNIDAD = VAL (ORDEN$)
80 PRINT LEFT$ (N$,20), UNIDAD
90 SUMA = SUMA + UNIDAD
100 GOTO 30
110 MEDIA = SUMA / N
120 PRINT "MEDIA DE UNIDADES SOLICITADAS";MEDIA
130 CLOSE #1
199 END
```

Como el nombre del cliente y el número de unidades adquiridas sólo figuran en un campo, es preciso separar las dos partes de ese campo.

Pepe Luis García, 25

Campo N$

Para ello se utiliza la línea 60 (el nombre se supone tiene una longitud máxima de 20 caracteres y el número 3 dígitos) que extrae la subcadena derecha, número de unidades. La línea 70 convierte la subcadena en un número decimal. La línea 80 escribe el *nombre del cliente* (subcadena izquierda y el *número de unidades* adquiridas. La línea 120 escribe la media de unidades adquiridas diariamente.

4. *El fichero MUESTRAS contiene los resultados sucesivos de las temperaturas obtenidas a lo largo de una serie de días con tres termómetros en una determinada fábrica. Se desea visualizar estas temperaturas y su media, y a continuación la temperatura media total de la muestra.*

   El proceso a seguir podría ser el siguiente:

   1. Abrir el fichero como entrada.
   2. Imprimir cabeceras.
   3. Leer el fichero completo:
      - Leer un registro X,Y,Z.
      - Si X = −999 terminar el bucle.
      - Calcular la media y suma parciales.
      - Imprimir dato.
   4. Calcular e imprimir media total.
   5. Cerrar el fichero.

   El programa podría ser el siguiente:

```
10 OPEN "MUESTRAS" FOR INPUT AS #1
20 SUMA = 0
30 PRINT "TEMPERATURA 1","TEMPERATURA2","TEMPERATURA3","MEDIA"
40 INPUT#1,X,Y,Z
50 IF X = -999 THEN 110
60 N = N+1
70 MEDIA = (X+Y+Z) / 3
80 SUMA = SUMA + MEDIA
90 PRINT X,Y,Z,MEDIA
100 GOTO 40
110 MEDIATOTAL = SUMA/N
120 PRINT,"LA MEDIA TOTAL ES: ";MEDIATOTAL
130 CLOSE#1
199 END
```

5. *El siguiente programa crea o lee un fichero secuencial de alumnos de una clase, que contenga en cada registro los nombres y apellidos, las notas orales obtenidas (3), el examen, el proyecto y la media de la asignatura (suma 5 notas divida por 5).*

```
10 REM "SEQ1"
20 INPUT "CREACION (0)/ LECTURA (1):";X
30 IF X=1 THEN 170
40 INPUT "NOMBRE DEL FICHERO :";X$
50 OPEN "O",3,X$:REM "REGISTRO"
60 INPUT "NOMBRE :";N$
65 IF N$="FIN" THEN 160
70 INPUT "APELLIDO :";P$
80 INPUT "NOTA 1 :";A
90 INPUT "NOTA 2 :";B
100 INPUT "NOTA 3 :";C
110 INPUT "EXAMEN :";D
120 INPUT "PROYECTO :";E
130 F = (A+B+C+D+E)/5
140 PRINT #3,N$;",";P$;",";A;",";B;",";C;",";D;",";E;",";F
150 GOTO 60
160 CLOSE #3:END
170 INPUT "NOMBRE DEL FICHERO : ";X$
180 OPEN "I",3,X$:REM "LECTURA"
190 IF EOF(3) THEN 160
200 INPUT #3,N$,P$,A,B,C,D,E,F
210 PRINT N$,P$,A,B,C,D,E,F:FOR I=1 TO 2000:NEXT I
220 GOTO 190
```

6. *El siguiente programa crea el fichero de acceso directo del programa 5.*

```
10 REM "DIRECT1"
20 INPUT "NOMBRE DEL FICHERO: ";X$
30 OPEN "R",#1,X$,32
40 FIELD #1,10 AS N$, 10 AS P$, 2 AS A$, 2 AS B$, 2 AS D$, 2 AS E$, 2
AS F$, 2 AS G$
50 INPUT "CREACION (0) / LECTURA (1) :";X
60 IF X=1 THEN 180
70 INPUT "NUMERO DE CODIGO (MAX. 3 CIFRAS) :";CODIGO%:REM "REGISTRO"
80 INPUT"NOMBRE :";NO$
90 IF NO$="FIN" THEN 240
100 INPUT "APELLIDO :";PN$
110 INPUT "A=";A:INPUT "B=";B:INPUT "D=";D
120 INPUT "E=";E:INPUT "F=";F:G = (A+B+D+E+F)/5
130 LSET N$=NO$:LSET P$=PN$
140 LSET A$=MKI$(A):LSET B$=MKI$(B):LSET D$=MKI$(D)
150 LSET E$=MKI$(E):LSET F$=MKI$(F):LSET G$=MKI$(G$)
160 PUT #1,CODIGO%
170 GOTO 70
180 INPUT "NUMERO DE CODIGO (MAX. 3 CIFRAS) :";CODIGO%:REM "LECTURA"
190 IF CODIGO%>999 THEN 240
200 GET #1,CODIGO%
210 PRINT N$:PRINT P$:PRINT CVI(A$):PRINT CVI(B$)
220 PRINT CVI(D$):PRINT CVI(E$):PRINT CVI(F$):PRINT CVI(G$)
230 GOTO 180
240 CLOSE #1:PRINT "FIN DE TRABAJO":END
```

PROGRAMAS PROPUESTOS

1. Crear un fichero secuencial del inventario de los artículos de una tienda de deportes. La información de cada registro debe contener los siguientes campos:

   — Número de filiación o código del artículo.
   — Nombre del artículo.
   — Número de unidades en almacén (Stuck).

2. Escribir un programa que permita introducir nombres y direcciones en un fichero secuencial.

3. Escribir un programa que lea nombres y direcciones de las personas que posean un determinado código postal introducido por teclado.

4. Un profesor ha almacenado los expedientes de sus alumnos en un fichero ESTUDIANTES. Cada registro contiene:

Nombre:	30 caracteres
Número del alumno:	5 dígitos
Calificación examen 1:	3 dígitos (7.50, 3.90, etc.)
Calificación examen 2:	3 dígitos
Calificación prácticas 1:	2 dígitos (6.8, 7.6, etc.)
Calificación prácticas 2:	2 dígitos
Calificación prácticas 3:	2 dígitos

   Escriba un programa que:

   a) Inserte las siguientes calificaciones:

   2 notas de exámenes.
   3 notas de prácticas.

   b) Calcule la media de las calificaciones de los exámenes y de las prácticas.
   c) Calcule la nota final de la asignatura de acuerdo a esta fórmula.

   Exámenes*.60 + Prácticas*.40

5. La empresa MANUFACTURAS R2D2 tiene los siguientes datos de ventas en su división JUEGOS, en un fichero denominado JUEGOS.DAT:

Año	*Ventas*
1980	12.500.000
1981	11.799.100
1982	13.600.000
1083	13.900.000
1984	14.100.000
1985	15.100.000

   Escribir un programa para almacenar estos datos en un fichero secuencial.

6. Escriba un programa que le permita introducir nombres y direcciones de una agenda en un fichero secuencial.

7. Escriba un programa que le permita introducir nombres y direcciones en un fichero de acceso directo. Debe utilizar un registro índice que contenga el número total de nombres y direcciones.

8. Se desea construir un fichero de datos de acceso secuencial denominado «CLASE» con 25 registros. Cada uno de ellos debe tener los siguientes campos: Apellidos, Nombre, Nota 1, Nota 2, Nota 3, Examen, Prácticas y Media.

   [(Nota 1 + Nota 2 + Nota 3 + Examen + Prácticas)/5 = Media]

CAPITULO 12

Puesta a punto de los programas

0. INTRODUCCION

La ejecución de un programa de modo satisfactorio suele llevar consigo un gran número de horas de trabajo, que pueden acortarse o alargarse en función del método de trabajo que se ha llevado.

Desde que se planifica la tarea de realización de un programa hasta su puesta a punto final se ha de pasar por diferentes fases cuyas ejecuciones correctas permitirán el éxito del mismo.

Se puede dividir un programa en dos partes bien diferenciadas.

— El análisis del problema, que llevará consigo el método adoptado para su resolución.
— La realización del programa en la computadora.

La complejidad de la fase de análisis será mayor o menor en función del planteamiento del programa y de los resultados que se esperan. Lo normal en esta fase será realizar un organigrama de las tareas generales que ha de ejecutar el programa. En el mencionado organigrama es preciso indicar en cada bloque: las variables de entrada, la tarea que realiza el bloque, y todas las entradas-salidas. Normalmente se incluyen comentarios en el programa.

Recordemos los símbolos más utilizados en los organigramas.

1. Comienzo/Fin (1)
2. Operaciones (2)
3. Comoparaciones (test) (3)
4. Entradas/salidas general (4)
5. Entrada/salida por teclado (5)
6. Subrutinas (6)
7. Disco (7)
8. Disquete (8)
9. Impresora (9)

En BASIC la puesta a punto de los programas es relativamente fácil, cualquiera que sea el sistema, dado que la ejecución de un programa es interactiva.

El proceso de *puesta a punto o corrección del programa* («debbugging») es fundamental en el programa de aplicación y, pese a las grandes ayudas del intérprete BASIC, deberá prestar especial atención al mismo.

El intérprete BASIC detectará los errores de sintaxis así como el número de las líneas en que se produjo. Si el error fuese de ejecución, se interrumpiría la ejecución con un mensaje de error de la sentencia que lo produjo. Sin embargo, puede ocurrir que en este último caso la interpretación del mensaje de error sea difícil y no se pueda detectar fácilmente la causa del error; entonces puede ser necesario ir ejecutando por partes el programa, incluyendo sentencias STOP o END provisionales, así como PRINT para obtener resultados parciales hasta conseguir localizar la sentencia que provocó el error. Inmediatamente después de su detección se procede a eliminar las sentencias provisionales introducidas.

Algunas computadoras tienen unas posibilidades de puesta a punto adicionales que permiten obtener la impresión o visualización de las interrupciones que se están ejecutando, con lo cual es mucho más fácil detectar el error. Esta ejecución llamada «paso a paso» ya se realiza mediante *órdenes* (comandos) del tipo TRON/TROFF, que permiten la ejecución del programa línea a línea.

Existe la posibilidad, en algunos sistemas, de situar *puntos de interrupciones* («breakpoints») en diferentes puntos del programa a fin de comprobar lo que está sucediendo en todo momento y reemprender la ejecución, eventualmente, paso a paso (línea a línea).

Otra posibilidad que se utiliza con frecuencia sería insertar o borrar las líneas que se considere que puedan influir en el desarrollo del programa. La numeración de líneas de 10 en 10, 100 en 100, etc., facilita considerablemente la puesta a punto y tal vez esta característica sea una de las que más influyen en la buena realización de un programa.

Es muy importante que el programador realice un estudio detenido del programa con papel y lápiz antes de lanzarse a la aventura de introducirlo en la computadora. Los lenguajes del tipo intérprete tienen el peligro de que un mal análisis previo puede demorar considerablemente el programa. Si cuando ejecute el programa en la primera ocasión no funciona, no debe intentar su ejecución de nuevo hasta que se haya realizado un análisis de las posibles causas de error. Es conveniente en este caso realizar un listado parcial o total del programa estudiándolo si fuera preciso con lápiz y papel. La ayuda de una impresora favorece positivamente el proceso de puesta a punto de los programas.

A la terminación de este capítulo le daremos las recomendaciones más usuales y prácticas para:

— Escribir un programa.
— Numerar las líneas.
— Ejecutar un programa.
— Depurar un programa.
— Ahorrar memoria.
— Aumentar la velocidad de su programa.

Asimismo le daremos unos consejos de interés general que pueden serle de utilidad en su labor diaria frente a la computadora.

1. EDICION DEL PROGRAMA

Ya se vieron en los capítulos 1 y 2 los procedimientos de edición más usuales en las computadoras y que permitirán modificar una línea de programa sin escribirla completamente. Dado que cada computadora suele recurrir a un sistema diferente, le aconsejamos que estudie con detenimiento el manual correspondiente. Haremos una revisión de las características más sobresalientes de los editores más frecuentes.

Existen principalmente *dos métodos de edición,* según se trabaje sobre la zona o memoria pantalla, o bien sobre la zona programa directamente. Las órdenes y las modificaciones se ven siempre sobre la pantalla, pero en unos casos se trabaja directamente sobre la zona programada

(TRS-80, DRAGON-64, Spectrum, etc.) y en otros la modificación se hace sobre la línea visualizada en pantalla, o sea la memoria pantalla de modo que la pulsación de la tecla RETURN (o similar) envía la línea modificada a la memoria del programa (éste es el caso de las computadoras de COMMODORE, IBM/PC, Amstrad, etc.

El método de edición directa en zona de programa lo tienen adoptado BASIC Microsoft y numerosas computadoras. Posee órdenes muy potentes que facilitan las modificaciones mediante la sustitución de pulsación de teclas por palabras claves. El método de edición en zona pantalla requiere la posibilidad del movimiento del cursor en los cuatro sentidos, a lo largo de la pantalla, y teclas para inserción y borrado de líneas/caracteres.

Ambos métodos tienen ventajas e inconvenientes, y así cada fabricante es fiel a su procedimiento perfeccionándolo en sus sucesivos modelos. El método de edición directa tiene la *ventaja* de sus órdenes potentes y el *inconveniente* de ser utilizable sólo en el modo de edición (EDIT o EDIT LINE), y, si necesita movimientos de cursor, tendrá que simularlos mediante el código ASCII o correspondiente a través de CHR$. El método de edición en zona pantalla presenta la *ventaja* de que los movimientos de cursor son utilizables en la ejecución de programas directamente y permiten la composición de dibujos en la pantalla; sin embargo tienen el inconveniente de exigir teclas especiales para la edición y que las modificaciones exigen más manipulaciones de las mismas.

Actualmente, la mayoría de los fabricantes incluyen otras posibilidades de edición como: *renumeración de líneas* (RENUMBER), *numeración automática* (AUTO), *copia de pantalla* (COPY), e incluso la posibilidad de insertar subprogramas que han sido realizados aparte.

Algunas computadoras permiten que programas auxiliares o «correctores» realicen las tareas citadas; sin embargo creemos que excepto en algunos casos no son muy rentables, en relación a sus precio.

2. NUMERACION DE LINEAS

Todos los programas escritos en BASIC deben tener sus líneas numeradas. El número de línea indica que las instrucciones expuestas después del mismo forman parte de un programa que se está escribiendo (introduciendo en memoria) o modificando. Si no existiera el número, las instrucciones se interpretarían en modo directo o inmediato y se ejecutarían nada más pulsar la tecla RETURN.

Los números de líneas se deben clasificar en orden creciente y, usualmente, se numeran de 10 en 10 para permitir la introducción de nuevas líneas.

Si su intérprete de BASIC permite varias instrucciones en una línea, deberá recurrir a este procedimiento siempre que tenga problemas de memoria o, lo que es igual, su programa ocupe casi toda la memoria libre para el usuario. Si no tiene limitaciones de memoria, el poner varias instrucciones en una sola línea disminuye la legibilidad del programa.

Además de las razones anteriores, la numeración de las líneas permite las bifurcaciones o rupturas de secuencias en la ejecución de los programas mediante sentencias GOTO, GOSUB, etc. Existen intérpretes de BASIC potentes (HP 9845, VAX de Digital, ANSI BASIC, etc.) que permiten la bifurcación a etiquetas «palabras» en vez de a etiquetas «números»; por ejemplo, GOSUB *Newton* en vez de GOSUB 10000, en donde *Newton* es la etiqueta de la dirección 1000.

Para facilitar la numeración de las líneas se dispone de las órdenes AUTO (la computadora escribe automáticamente el número de línea con el consiguiente ahorro para el usuario) y RENUMBER (remunera parte o todas las líneas de un programa).

3. PUESTA A PUNTO DE LOS PROGRAMAS

La puesta a punto o depuración de un programa es el proceso de ejecución/corrección de errores, de modo que a su terminación estará preparado para ser utilizado de modo continuo sin que plantee problemas su ejecución. Dicho de otro modo, la *depuración* es el proceso de búsqueda y eliminación de errores.

Desgraciadamente y a pesar de sus esfuerzos, un programa contendrá errores la primera vez que se ejecute. Los errores de programas se denominan en inglés «bug» (pulgas) y el proceso de localización y corrección, *depuración* o *puesta a punto.*

Sin embargo, es preciso advertir que incluso la ejecución del programa hasta su terminación sin errores, no significa que el programa funcione correctamente. Significa simplemente que no se han encontrado errores de sintaxis y que se han utilizado sentencias válidas de la versión BASIC que utiliza. Existen numerosas causas por las que un programa puede dar la falsa sensación de funcionar bien: errores de redondeo que sólo salen con números grandes, operaciones numéricas o de cadenas en casos extremos, etc.

La etapa de puesta a punto de un programa suele durar normalmente tanto tiempo como la escritura, y en ocasiones más.

Como ya se ha comentado, normalmente un programa no funcionará *a la primera;* sin embargo, si ha tenido la precaución de escribirlo claramente, incluyendo comentarios oportunos en diferentes partes del mismo, la puesta a punto del programa se conseguirá rápidamente.

Los lenguajes intérpretes, como BASIC, facilitan considerablemente la puesta a punto de los programas, ya que permiten la corrección de errores a medida que se introducen por teclado o ejecutan mediante RUN.

Si los resultados de un programa no son los esperados, es preciso hacer un recorrido o *rastreo* del mismo desde la línea inicial siguiendo el camino que recorre la computadora en su ejecución. Se facilita esta tarea intercalando sentencias STOP en puntos claves o bien PRINT para ir viendo los valores que toman las diferentes variables. *Recuerde* que un programa interrumpido con STOP se puede continuar mediante la orden CONT. Posteriormente le diremos cómo detectar y corregir los errores más frecuentes.

Como ya se ha visto en el Capítulo 10, existen un conjunto de sentencias/órdenes y funciones que permiten detectar o capturar errores en su ejecución, así como visualizar el contenido de diferentes variables en el desarrollo del programa.

Deberá incluir en su programa, al principio, la sentencia ON ERROR GOTO y posteriormente bifurcaciones condicionales mediante las funciones/variables ERR y ERL que permitan detectar el número de línea donde se produjo el error, así como el código del mismo, con los que podrá deducir fácilmente el error y el motivo que lo produjo, corrigiendo el programa en consecuencia.

A lo largo de un programa, si se comprueba que una determinada variable no funciona correctamente, será preciso comprobar el valor que tiene a lo largo de su utilización. Para ello deberá recurrir a imprimir su valor en los puntos críticos mediante sentencias PRINT o bien de modo continuo con la orden/sentencia DSP.

Si pese a los procedimientos anteriores no logra averiguar el error que tiene su programa, deberá *rastrearlo* o, lo que es igual, ejecutarlo línea a línea mediante la orden TRON (o TRACE); de este modo podrá comprobar el/los número/s de línea/s donde se presentan dificultades. Una vez que haya deducido los errores y el procedimiento de corregirlos, desactive la función TRON mediante TROFF (NOTRACE).

Algunos programadores tienen por norma ejecutar el programa paso a paso mediante TRON, la primera vez que se «rueda» el mismo, a fin de comprobar la «bondad» o «fiabilidad del mismo.

La interpretación de los principales tipos de errores es un problema importante, y no siempre el laconismo del mensaje permite su adecuada interpretación y, por consiguiente, su corrección. Se van a dar unas reglas para prevenir al lector sobre los errores más frecuentes y el modo de investigarlos si se producen. Agruparemos los errores en tres grupos: *de sintaxis, de ejecución y de lógica.*

3.1. Errores de sintaxis

Estos errores son los más fáciles de encontrar.

En primer lugar examinar cada elemento de la línea:

— Número de la instrucción.
— Las palabras claves (sentencias, funciones, operadores, etc.) están bien escritas.
— Los nombres de las variables y constantes son correctos.
— Los separadores (espacios en blanco, comas, puntos y comas, dos puntos...) están escritos y situados correctamente.

En segundo lugar estudiar la estructura de las instrucciones:

— En las expresiones aritméticas (ver si faltan o sobran paréntesis, separadores, etc.).
— Comprobar que no se mezclan variables numéricas y de cadena.
— Verificar en las sentencias de comprobación (IF-THEN...) si los operadores lógicos están bien situados.
— Examinar las instrucciones que siguen a THEN.
— Verificar las expresiones utilizadas y sus valores iniciales, final y paso en los bucles FOR-NEXT.
— Comprobar que cada sentencia NEXT está asociada con su variable de control y corresponde con su FOR.

En la entrada/lectura/salida de datos:

— No confundir variables numéricas con variables de cadenas y viceversa. Su tratamiento es diferente.

En las sentencias GOTO/GOSUB/ON- GOTO/ON-GOSUB:

— Verificar que existen todos los números de líneas a donde se bifurcan.

De modo general:

— Verificar las variables de subíndice y comprobar si están dimensionadas adecuadamente.
— Comprobar que a cada sentencia READ le corresponde un DATA o bien existe la sentencia RESTORE correspondiente.

3.2. Errores de ejecución

Estos errores se conocen en la jerga informática como *«run time»*. Se pueden distinguir dos casos: *a)* El programa no se ha ejecutado nunca, y *b)* el programa ya ha funcionado.

a) *El programa no se ha ejecutado nunca*

Si el *programa no se ha ejecutado nunca,* puede tratarse de un error lógico a nivel de análisis o escritura del programa:

— Si no se observa error en ningún resultado se deben comprobar los bucles.
— Si los resultados son falsos se ha de tratar de localizar dónde se produjo el error.
— Si se trata de cálculos, será preciso verificar si las expresiones aritméticas son correctas, especialmente si se cumple la jerarquía de operadores.
— Si el programa realiza lo contrario de lo previsto, hay que verificar que las condiciones de las pruebas (test) están bien definidas.
— Verificar posibles errores de entrada-salida.
— Si utiliza bucles FOR-NEXT compruebe que los valores iniciales, finales y pasos de cuenta son correctos. Examinar cada pareja FOR-NEXT y comprobar que no se ha olvidado NEXT o su variable asociada.
— Comprobar, mediante sentencias PRINT adicionales, cuáles son los resultados falsos.
— Examinar el contenido de las variables que ofrezcan dudas sobre su funcionamiento.
— Comprobar que todas las subrutinas terminan con RETURN.

Como *regla general* se deberán analizar con detenimiento los mensajes de error que aparecen en pantalla; en muchos manuales de usuario, además del mensaje, se suelen dar sugerencias para solucionar el error.

b) *Errores después del funcionamiento*

Si *el programa ya ha funcionado* los posibles errores que se pueden encontrar después de varias ejecuciones se deben esencialmente al hecho que los juegos de datos varían de una ejecución a otra.

Posibles causas de error son:

— Datos particulares que tienen el control sobre partes del programa pueden no haberse utilizado todavía y, por ello, aparecen errores de programación aún no detectados.
— En algunos casos, estos datos particulares, o alguno de sus valores, no han sido previstos y provocan errores en la ejecución cuando se producen; por ejemplo, «desbordamiento», «división por cero», etc.

Es preciso prever en el programa todos los conjuntos de datos así como los posibles valores particulares, sobre todo los menos frecuentes, a fin de evitar errores cuando se produzcan. Los resultados imprevistos, si no se han previsto, pueden originar enojosos errores.

Otro motivo de error reside en el desarrollo del programa. Normalmente se ejecuta por personas distintas a las que lo construyen, por lo que si los mensajes y preguntas que aparecen durante su ejecución no son lo suficientemente claros, pueden ser mal interpretados por el usuario y, por consiguiente, proporcionar errores que una buena manipulación evitaría. Existe una teoría entre algunos programadores que aconseja antes de poner en servicio un programa ponerlo en manos de personas no experimentadas para comprobar la «bondad» y «robustez» del programa. Si cualquier usuario de un programa no es capaz de ponerlo en marcha sin manipulación errónea del mismo, el programa se considera que no está puesto a punto.

A lo largo del programa se debe guiar al usuario de tal forma que cuando cometa errores de manipulación éstos puedan ser detectados y corregidos. Las sentencias ON ERROR GOTO dentro del programa facilitan esta tarea.

Tabla 12.1. Mensajes de error típicos en ejecución de programas.

Mensaje	*Posibles causas*
NEXT whitout FOR	Se ha encontrado una sentencia NEXT sin su correspondiente FOR.
FOR whitout NEXT	Se ha comenzado un bucle FOR y no se encuentra su correspondiente NEXT.
RETURN whitout GOSUB	Se ha encontrado una sentencia RETURN sin su correspondiente GOSUB.
WHILE without WEND	Un bucle WHILE no se termina con su correspondiente WEND.
WHEND without WHILE	Se he encontrado una sentencia WHEND sin su correspondiente WHILE.
Syntax error	Error de sintaxis en una línea de programa.
Out of data	Datos agotados en sentencias READ-DATA.
Subscript out of range	Subíndice de una matriz fuera de rango. Se ha utilizado una variable con índice superior a la dimensión fijada.
Undefined line number	Número de línea no definida. Se ha transferido el control del programa a un número de línea que no existe.
Type mismatch	Tipos no coincidentes. Se ha utilizado una variable o constante de cadena donde se requería una variable o constante numérica o viceversa.
Division by cero	Se ha ejecutado una operación por cero.
Illegal function call	El argumento de una función BASIC no es legal o está fuera de rango.

Una lista de los mensajes de error típicos que suelen aparecer al ejecutar programas y sus posibles causas se enumeran en la tabla 12.1.

3.3. Errores en la lógica del programa

Los errores en la lógica del programa son los más difíciles de detectar y de corregir, ya que cuando se produce un error de este tipo, el programa, normalmente, se ejecuta hasta el final aunque «lógicamente» la salida será incorrecta.

Los errores de lógica se eliminan con una buena planificación. La división en modelos independientes o en subrutinas que ejecuten tareas específicas contribuye en gran medida a disminuir los errores. Es muy usual dividir un programa en los siguientes modelos:

a) Lectura/Entrada de datos:

- de teclado y/o de ficheros en sentencias DATA o en unidades de casete o disco.

b) Proceso de datos.
c) Actualización de datos.
d) Presentación de resultados:

- en pantalla,
- en impresora.

Una buena estructura del programa en módulos independientes (con el menor uso posible de la sentencia GOTO) revierte *siempre* en el buen funcionamiento del programa.

Como regla práctica conviene resaltar que existen sentencias cuya programación lógica exige notable atención: GOTO, IF-THEN, FOR-NEXT, WHILE-WEND, etc. Veamos algunos ejemplos prácticos de malas programaciones.

1.
```
100 A = B+C : GOTO 600 : PRINT "Demo"
```

La sentencia PRINT «Demo» no se ejecutará nunca, ya que el control del programa saltará *siempre* a la línea 600.

2.
```
100 FOR N=1 TO 50 STEP -1
```

El bucle no se puede ejecutar correctamente la que exige un incremento en STEP, positivo y no negativo.

3.
```
100 IF N=10 AND N=100 THEN 500
```

La condición de la línea 100 no se cumplirá nunca (N no puede ser, a la vez, 10 y 100); por consiguiente el programa nunca bifurcará a la línea 500.

4.
```
100 IF N >30 OR N>50 THEN PRINT "Demo"
```

Nunca se escribirá «Demo» ya que N no puede ser mayor que 30 o menor que 50 a la vez.

5.
```
100 N=7
110 WHILE N => 7
120     PRINT N*5
130     PRINT N^2
140 WHEND
150 ...
```

El bucle WHILE-WEND no se ejecutará nunca, ya que el valor de N es igual a 7 en la línea 100.

Una técnica muy útil para desarrollar un programa es utilizar sentencias PRINT entre los módulos que lo componen. Una vez terminado el programa y verificadas las salidas se pueden eliminar las sentencias PRINT adicionales.

3.4. Reglas prácticas para la depuración de programas

El problema esencial en la depuración de un programa es aislar la fuente que produce el error *(bug)*. Aunque ya se han comentado, en este capítulo y en el 10, los métodos usuales empleados en la depuración, vamos a recordarlos con algunos ejemplos prácticos:

Sentencias/Ordenes PRINT, STOP y CONT

La ayuda más sencilla a la depuración es la simple combinación de las sentencias PRINT-STOP. Supongamos que la variable X tiene un valor correcto en la línea 680 del programa, pero es incorrecta cuando se imprime en la línea 910. Tras examinar el programa, sospecha que la línea 850 es la fuente del error.

```
680 ...
850 ...
910 ...
```

Se puede probar añadir las siguientes líneas de prueba:

```
100 LPRINT "680 ..."
120 LPRINT "850 ..."
130 LPRINT "910 ..."
845 PRINT X
855 PRINT X :STOP
```

que imprimen el valor de X antes y después de la línea sospechosa 85. La sentencia STOP detiene el programa para inspeccionar ambos valores de X. Si X es correcto antes de la línea 845 e incorrecto en 855, el error está en la línea 850. Si X fuese correcto en 855, se inserta otra pareja PRINT-STOP más adelante, por ejemplo:

```
895 PRINT X :STOP
```

Para continuar la ejecución, utiliza la orden CONT.

Impresión en modo directo

Mediante PRINT actuando como orden se puede comprobar el valor de las diferentes variables, o los contadores de los bucles, tras la ejecución del programa.

Sentencias TRON, TROFF

A veces es difícil seguir el flujo del programa, en particular si existen numerosas sentencias de bifurcación o el programa es largo. En esos casos es conveniente, y en ocasiones imprescindible, la ejecución paso a paso para comprobar el flujo correcto así como los resultados parciales y totales.

La activación del rastreo o traza de programa se suele hacer como orden o como sentencia dentro del programa. Personalmente preferimos como orden, ya que de este modo se independiza el rastreo de la ejecución del programa.

Captura de errores

La captura de errores es, sin duda, uno de los procedimientos más seguros y recomendables para una buena puesta a punto de un programa. En el Capítulo 10 se explicaron detenidamente las sentencias ON ERROR GOTO, ERROR, ERR, ERL y RESUME.

4. REGLAS PARA LA BUENA REALIZACION DE UN PROGRAMA

Se puede deducir de todo lo expuesto anteriormente que *programar bien* no es solamente escribir un programa que «funcione», sino escribirlo previendo todo cuanto pueda suceder, y la posibilidad de modificarlo y adaptarlo a las necesidades del usuario lo más rápidamente posible.

La marcha a seguir para escribir en programa suele convertirse al final en un método adaptado a cada usuario. Sin embargo, el aprender bien el lenguaje de programación, reflexionar sobre el problema a resolver y dibujar o diseñar un organigrama o diagrama de flujo son tres características o etapas esenciales para construir buenos programas.

Suponiendo que ya conocemos bien el lenguaje, en nuestro caso BASIC, vayamos a la segunda etapa. Un gran peligro consiste en precipitarse sobre la máquina y comenzar a martillear «furiosamente» el teclado antes de reflexionar seriamente sobre la tarea a realizar. Es preciso dividir el problema en una serie de etapas ejecutables independientemente, obteniendo lo que se llama *un algoritmo.* La conversión de este algoritmo en un buen programa *(codificación)* será fácil si se ha reflexionado el tiempo necesario sobre las etapas que constituyen cada algoritmo, y que ahorrarán mucho tiempo de depuración. Una vez que ya ha comenzado la codificación y su introducción por teclado consideraremos algunas reglas para una buena programación en BASIC.

4.1. Reglas para escribir un buen programa en BASIC

- Dar nombre a su programa, con lo que se facilitará su búsqueda en la unidad de casete o disco.
- Los números de línea autorizados *suelen* ir de 0 a 65529.
- Las líneas de programa se pueden *introducir* en un orden cualquiera.
- Cada línea de programa no debe exceder de 255 caracteres.
- La abreviatura ? suele representar en muchas versiones BASIC a la sentencia PRINT.
- Explicar a lo largo del programa y sobre las líneas de comentarios — sentencias REM, abreviatura (') o (!)— lo que hace y cómo lo hace.
- Elegir nombres de variables del tipo nemotécnico en la medida de lo posible y que permita recordar fácilmente su función.
- Las variables esenciales de un programa deben ir en cabeza.
- Separar y rotular mediante REM las diferentes partes de un programa.
- Descomponer el programa en subrutinas, definiendo si es preciso las variables de entrada y salida.
- Poner comentarios en los puntos esenciales.
- Poner sangría en las sentencias interiores de los bucles. FOR-NEXT (a lo largo del libro no se ha hecho normalmente, pero se lo recomendamos expresamente en este punto).

```
100 FOR I=1 TO 100
110       PRINT I, I^2
120       PRINT
130 NEXT I
```

- Evitar la evaluación repetida de expresiones:

```
100 X = A*5+B                          110 M = A*5+B + Y
110 M = X+Y          en lugar de       120 N = A*5+B + Y
120 N = X-Y
```

- Utilizar funciones incorporadas del sistema en lugar de operaciones.
- Utilizar IF THEN e IF THEN ELSE, que evite las sentencias GOTO; tal vez sea ésta una de las recomendaciones fundamentales: evitar la utilización de la sentencia GOTO.
- Si sitúa varias sentencias sobre una sola línea tienen que ir separadas mediante separadores (:, @, etc.).

4.2. Ejecución del programa

La ejecución del programa comienza con la orden RUN, que inicializa a cero todas las variables del programa.

Se puede ejecutar un programa también mediante GOTO en modo orden, es decir, sin incluir en programa, con lo que éste comienza a ejecutarse en la línea indicada por GOTO. Se diferencia de la orden RUN en que no se ponen a cero las variables.

El programa se puede detener en cualquier momento mediante la orden BREAK o STOP. Sin embargo, hay ocasiones en que la computadora entra en un bucle sin fin y aun pulsando BREAK o STOP el programa no se detiene, por lo que será preciso recurrir al botón pulsador RESET. Igual sucede si el programa ha llamado a una subrutina en lenguaje ensamblador o código máquina.

Un programa detenido mediante STOP/BREAK puede continuar su ejecución mediante la orden CONT. La puesta a punto del programa se facilita considerablemente con las órdenes STOP-CONT.

4.3. Fallos en el programa

En un buen programa se deben prever todas las posibles contestaciones que un usuario puede dar ante una petición de datos. Así, por ejemplo, si el programa pide a un usuario introducir datos, es preciso que dicho programa admita todas las posibles soluciones aunque algunas no sean frecuentes. Por ejemplo, en el caso de una entrada de la edad de una persona, no es usual introducir 0 o −5; sin embargo, si el usuario comete un error al introducir el dato y el programa no ha previsto este caso, nos originará un indeseado mensaje de error; es preciso estar preparado para esta eventualidad.

Un buen programa, sobre todo si es profesional y va a ser utilizado por personas distintas a quienes lo realizaron, no debe producir fallos o «averías» relevantes. Los fallos o errores típicos de un programa se engloban dentro de tres grupos:

a) Fallo a nivel de sistema. Se pierde el control del sistema a través del teclado. Es debido en muchos casos a sentencias POKE que se interfieren con el programa monitor al introducirse en zonas de memoria peligrosas. El remedio suele ser reinicializar el sistema con la tecla o función RESET. Suele provocar una situación catastrófica para el programa, ya que como mínimo origina la iniciación del mismo.

b) Fallo a nivel de órdenes. No suele ser tan grave como el fallo anterior, pero sí muy enojosa. Presupone que la computadora abandona el programa y vuelve al nivel de órdenes. Algunas posibles causas son:

1. Pulsación no deseada de la tecla STOP.
2. Pulsación de RETURN en repuesta a una petición de datos, sin haber introducido antes los datos solicitados. (La pulsación de RETURN a una petición de una sentencia INPUT, supone introducir en la variable correspondiente un cero a una cadena nula.)

```
10 input n
20 print n

RUN
?                (pulsación RETURN)
0
```

```
10 INPUT N$
20 PRINT N$

RUN
?                (pulsación de RETURN)
0
```

c) Fallo con mensaje de error.
En el párrafo 3 se vieron las diferentes causas que provocan mensajes de error.

5. DESBORDAMIENTO DE MEMORIA

Cuando la memoria central de una computadora se ha superado por un programa, bien en modo elástico (introducción del programa por teclado) o bien de modo dinámico (al ejecutarse), nos aparecerá en pantalla un mensaje de error «OUT OF MEMORY» que significa *falta memoria o no queda memoria.* En estos casos será preciso reestructurar el programa para ahorrar espacio, o de lo contrario no se podrá ejecutar el mismo.

Para saber la memoria libre para el usuario, en cualquier momento deberá realizar la instrucción PRINT MEM o PRINT FRE *(argumento),* según el tipo de computadora.

Si a pesar de la reestructuración del programa sigue obteniendo el mensaje OUT OF DATA, no lo quedará más remedio que ir a una ampliación de la memoria por «hardware». Hoy casi todos las computadoras tienen en su configuración física conectores que permiten la expansión de memoria de los mismos. Si a pesar de ello a sus programas les sigue faltando memoria, deberá pensar en una computadora con mayor capacidad de memoria. Como norma general le podemos decir qué memorias centrales de 32K octetos a 64K octetos permiten un gran campo de aplicaciones y numerosísimos programas.

5.1. Información sobre la asignación (ocupación) de memoria

Para calcular el espacio de memoria ocupado por un programa será preciso, como ya se ha dicho, ejecutar PRINT FRE (1) o PRINT MEM. Sin embargo, es necesario que con anterioridad al desarrollo del programa se pueda realizar un cálculo aproximado sobre el espacio ocupado; para ello cada computadora presenta una ocupación de memoria, de sus variables, cadenas, etc., que si bien es similar no siempre es igual.

Aunque la mayoría de las computadoras suelen tener reglas muy similares de ocupación de memoria, no siguen una ley general, por lo que, en su caso, deberá comprobar los datos que el fabricante indica en su manual. Por desgracia no siempre da la información. En ese caso le aconsejamos que tome los datos que aquí le damos, pero con la debida reserva.

Así pues, si desea conocer el espacio aproximado de memoria ocupado por su programa deberá realizar los cálculos siguiendo estos criterios, tomados como modelo estándar, aunque podría ocurrir que su computadora actuase de diferente modo.

— *Las palabras claves* tales como FOR, GOTO, NEXT, etc., y los nombres de las funciones COS, INT, FIX, etc., necesitan un octeto de almacenamiento.
— *Los caracteres* de cualquier tipo utilizan un octeto por carácter.
— Una línea de programa requiere 5 octetos:

Número de línea 2 Puntero de línea 2 Retorno de carro 1

— Al definir una función por medioi de la *sentencia DEF* se utilizan 6 octetos para almacenar la definición.
— *Las variables numéricas o cadenas de caracteres simples* ocupan 7 octetos, 2 para el nombre y 5 para el valor.
— *Las variables simples de cadena* ocupan:

7 octetos: 2 para el nombre, 1 para la longitud, 2 para el puntero y 2 no se utilizan.

Además, cada carácter de la cadena ocupa un octeto.

— *Las variables subscritas de las matrices* ocupan 7 octetos para almacenar el encabezamiento, más los octetos que almacene cada elemento de la matriz. Si el elemento es una variable de cadena de caracteres requiere 3 octetos, si es de coma flotante requiere 5 octetos y si es entero 2 octetos.

Cuando un programa se ejecuta, *asignación dinámica de memoria* en *la pila* (zona de memoria que el intérprete BASIC reserva para almacenar las direcciones de retorno de las bifurcaciones GOSUB, GOTO, FOR- NEXT, etc.), de la forma siguiente:

1. Cada bucle FOR-NEXT consume 22 octetos.
2. Cada GOSUB activo (no terminado) utiliza 6 octetos.
3. Cada paréntesis de una expresión utiliza 4 octetos y cada resultado temporal, calculado de una expresión, utiliza 12 octetos.

Como orientación, en las tablas 12.2 y 12.3 se recoge la reserva de memoria que las microcomputadoras que funcionan con Microsoft BASIC (MBASIC, MSX BASIC Y BASICA del IBM/PC) y HP-85/87 emplean para sus variables simples o de matrices.

Tabla 12.2. Lenguajes BASIC Microsoft.

Variables simples	*Número de octetos ocupados*
Enteros	2 octetos
Simple precisión	4 octetos
Doble precisión	8 octetos
Cadenas	3 octetos, más longitud del contenido
Variables suscritas	*Número de octetos ocupados*
Enteros	2 octetos/elemento
Simple precisión	4 octetos/elemento
Doble precisión	8 octetos/elemento
Cadenas	3 octetos, más longitud del contenido

Tabla 12.3. Computadoras HP- 85/HP-87/BASIC ANSI.

Variables simples	*Número de octetos ocupados*
Entero	5 octetos
Simple precisión	6 octetos
Doble precisión	10 octetos
Cadenas	8 octetos+1 octeto por carácter
Variables subcritas	*Número de octetos ocupados*
Enteros	8 octetos+3 octetos por elemento
Simple precisión	8 octetos+4 octetos por elemento
Doble precisión	8 octetos+8 octetos por elemento
Cadenas	8 octetos+1 octeto por elemento

5.2. Normas para ahorrar memoria

La memoria empleada en un programa puede ser en algunos casos esenciales. Si el programa es largo y la memoria limitada, 32 o 64 kg por ejemplo, la economía es esencial. Sin embargo, si dispone de mucha memoria o bien el programa ocupa poco espacio, no merece la pena ahorrar memoria, ya que el esfuerzo no sólo puede ser valdio sino que, incluso, puede resultar perjudicial para la legibilidad del programa.

Para el caso de que necesite ahorrar memoria le damos a continuación unas normas para hacerlo:

1. *Escribir varias sentencias en cada línea.* Cada línea de programa utiliza 5 octetos. Dos de estos octetos contienen el número de línea en binario, 2 octetos el puntero de línea y 1 octeto el retorno de carro. Con independencia del número de línea, éste ocupará 2 octetos de memoria. Así pues, incluyendo en una línea varias sentencias se ahorran «octetos» de memoria a razón de 5 por línea suprimida.

2. *Eliminar los espacios innecesarios.* Por ejemplo:

   `10 PRINTX,Y,Z` *utiliza 3 octetos menos que* `10 PRINT X, Y, Z`

   Los espacios entre el número de línea y el primer carácter significativo no se tienen en cuenta.

3. *Eliminar las sentencias REM.* Cada comentario REM utiliza por lo menos 1 octeto de memoria, además de los necesarios para almacenar el texto. Por ejemplo:

   `10 REM Esto es un comentario` *ocupa 28 octetos de memoria.*

Si tiene problemas de memoria no utilice sentencias REM para documentar su programa.

4. *Utilizar variables en vez de constantes.* Si ha de utilizar la constante 3.141592 con frecuencia en el programa, añada la siguiente línea a su programa:

```
10 PI=3.141592
```

Una constante ocupa 4 octetos más que la variable. Si utilizó Pi, 20 veces en lugar de 3.141592, habrá ahorrado 80 octetos de memoria.

5. *Emplear números enteros en vez de números reales.* Las variables enteras se almacenan en 5 octetos, mientras que las variables reales se almacenan en 7. Si no necesita variables reales procure evitarlas.

6. *No utilice las sentencias END.* Un programa no necesita terminar con END, por lo que puede eliminarla. Ahorra 7 octetos de memoria.

7. *Reutilizar las mismas variables.* Si una parte de un programa utiliza una variable para almacenamiento temporal de un resultado, utilice la misma variable posteriormente para cualquier otra variable que necesite.

   Si utiliza bucles FOR-NEXT imbricados (no anidados) emplee el mismo índice.

   Si a lo largo del programa se pide al usuario respuestas como «SI», «NO», etc., utilice siempre las mismas variables, por ejemplo A$, B$, etc.
   Estas medidas influyen negativamente en la claridad del programa, pero ahorran mucho espacio de memoria.

8. *Utilizar GOSUB.* Si existen partes de programa repetitivas, se deben convertir en subrutinas.

9. *Utilizar el elemento 0 de las matrices: listas o tablas.* Por ejemplo, A(0), B(0), C(0), etc. Normalmente, y así lo hemos hecho en numerosas ocasiones, en el libro se suelen despreciar estos elementos, por comodidad; no lo haga si tiene problemas de espacio.

10. *Dimensionar explícitamente todas las variables suscritas.* Se debe dimensionar, aunque las variables tengan menos de 10 elementos.

```
DIM A (4)        DIM A$ (7)        DIM A (4, 6)
```

11. *En ocasiones, utilizar FRE(1) o MEM como variables.* Insertadas en un programa, puede conseguir que se comprima al máximo el espacio ocupado por el programa hasta ese momento.

    El procedimiento puede ser asignar FRE(1) o MEM a una variable no utilizada.

```
Z1=FRE (1)
```

12. Utilizar sentencias INPUT y ficheros de datos (en cinta o disquete) en vez de elementos de asignación y sentencias DATA.

6. COMO AUMENTAR LA VELOCIDAD DE EJECUCION DE UN PROGRAMA

Aunque el lenguaje BASIC es muy popular, y principalmente por su facilidad de aprendizaje, tiene una gran limitación: *la velocidad de ejecución.* En numerosas ocasiones deberá desistir de ejecutar el programa en lenguaje máquina (ASSEMBLER) o de alto nivel, tal como FORTRAN o FORTH, o bien comprarse un BASIC compilador si su computadora lo permite.

Sin embargo, existen determinadas reglas que, siguiéndolas, hacen que se aumente la velocidad máxima de ejecución de los programas escritos en lenguajes BASIC. Se presentará, no

obstante, un inconveniente, y es que normalmente el aumento de velocidad lleva consigo que los programas se hacen menos «claros» o legibles. Por ello deberá estudiar un compromiso entre estas dos características.

La velocidad del programa suele disminuir cuando se realizan programas que contienen ecuaciones y funciones matemáticas complejas, o bien programas con bucles FOR-NEXT largos (como es el caso de los cálculos estadísticos). Muchos de los consejos que damos a continuación son comunes a los dados en el párrafo 5.2, por lo que en muchas ocasiones el ahorro en espacio de memoria lleva consigo un aumento en la velocidad de ejecución.

Las técnicas de optimización más usuales son:

1. *Eliminar todos los espacios innecesarios* así como las sentencias REM. Los espacios son caracteres que deben ser ignorados y su interpretación consume tiempo. Igual sucede con REM.
2. *Utilizar variables en lugar de constantes.* Esta es probablemente la sugerencia más importante para aumentar la velocidad. Se requiere más tiempo para convertir una constante a su representación en coma flotante que ir a buscar el valor de una variable simple o subscrita.
3. *Utilice, si es posible, nombres de variables de una sola letra.* El intérprete emplea tiempo en leer los restantes caracteres.
4. *Las variables utilizadas con frecuencia se ponen al principio del programa o de la línea en caso de que tengan varias sentencias.* Las variables se colocan en una «tabla de variables» en orden de asignación. Así, una línea 10 A=5:B=4:C=4 sitúa a A en primer lugar, a B en segundo y a C en el tercero. Posteriormente, cuando BASIC encuentre una referencia para buscar a la variable A, sólo tendrá que examinar una posición en la tabla, dos posiciones para B, 3 para C, etc.
5. *Utilizar las sentencias NEXT sin la variable de índice siempre que sea posible.* (Ya se sabe que hay computadoras que no lo admiten.) NEXT es algo más rápido que NEXT I ya que no se comprueba si la variable I especificada en el NEXT corresponde con la última sentencia FOR. Si aparece NEXT, BASIC supone que es correcta y lo asigna al último FOR.
6. *Las líneas a las que se llama con frecuencia se deben situar al principio del programa.* Por ejemplo al encontrarse una sentencia como GOTO 50, el programa inicia un recorrido por todas las líneas a partir de la primera hasta encontrar la línea 50. Por ello cuantas menos líneas se tengan delante se acortará el tiempo.
7. *Escriba las subrutinas al principio del programa.* La razón de la disminución es igual al párrafo anterior.
8. *En bucles FOR-NEXT utilizar variables enteras en lugar de variables reales o de coma flotante.*
9. *Numere las líneas de su programa de una en una.* Reduce los dígitos que se han de leer durante un GOTO o un GOSUB.
10. *Siempre que sea posible efectuar operaciones aritméticas antes de entrar en bucles:*

```
100 FOR A = 1  TO  N  STEP X-Y
110 S = SQR(F+G^3.6)*A
120 PRINT A,S
130 NEXT A
```

Como X-Y no se ve afectada por el valor del contador A, el realizar la operación X-Y fuera del bucle, evita calcular X-Y, N veces. De igual modo el cálculo de F+G13.6 se puede realizar fuera del bucle. Así, si el programa anterior se modifica de acuerdo con las ideas anteriores, se ejecutará a una velocidad mayor.

```
80  M=X-Y
90  B = F+G^3.6
100 FOR A=1 TO N STEP M
110 S=SQR (B) * A
120 PRINT A, S
130 NEXT A
```

11. *Eliminar mensajes de pantalla* innecesarios.

12. Como última norma si, pese a todo, ha llegado a un límite de velocidad (el BASIC no superará determinados tiempos) habrá de recurrir a *utilizar subrutinas en lenguaje máquina en los puntos críticos* (sobre todo en aquellas partes del programa que consumen tiempo de cálculo). El lenguaje máquina o ENSAMBLADOR (ASSEMBLER) se ejecuta a mucha mayor velocidad que el BASIC (en ciertos casos, se puede ganar un factor de 100). La mayoría de las computadoras que trabajan en BASIC permiten la programación en lenguaje máquina. Desgraciadamente esta solución es compleja, sobre todo para principiantes, por lo que si desea introducirse en el apasionante mundo del lenguaje máquina lea algún libro específico sobre lenguaje ensamblador en microprocesadores.

7. COMPATIBILIDAD DE PROGRAMAS EN COMPUTADORAS DIFERENTES

La *compatibilidad, portabilidad* o *transportabilidad* de un programa implica que se pueda transferir de una computadora a otra de marca y/o modelo diferente, y por consiguiente ejecutar sin ningún cambio.

En la actualidad la compatibilidad es el objetivo irreversible en la Microinformática de la segunda mitad de los 80. Sin embargo, excepto en muy contadas ocasiones, la compatibilidad absoluta es casi un mito; un programa que corre en la computadora X, muy difícilmente correrá en la computadora Y, utilizando el mismo casete o disquete y sin realizar cambios en el programa.

Las computadoras MSX tal vez sean la excepción que confirma la regla: un programa en casete o cartucho en la computadora Hit-bit de Sony correrá sin problemas en la computadora Philips MSX. Las computadoras IBM/PC y compatibles con las dos versiones de BASIC, GWBASIC y BASICA tiene compatibilidad casi total.

En la mayoría de los casos habrá que realizar cambios y en algunas ocasiones, si el sistema operativo en que corre el BASIC no es el mismo, incluso habrá que escribir por completo el programa, ya que la cinta de casete o disquete no serán compatibles.

El problema, dentro de la misma marca, sí ha sido resuelto por los fabricantes y así resulta que: un programa escrito para el ZX Spectrum, correrá bien en ZX Spectrum Plus y ZX Spectrum 128k; un programa escrito para el Amstrad 664 corre sin problemas en el Amstrad 6128; un programa escrito en BASICA para el IBM/PC corre en las computadoras IBM/XT y AT.

Por todo lo anterior si prevé que su programa será utilizado en computadoras distintas al modelo/marca en que lo ha escrito, le recomendamos lea y aplique detenidamente las reglas que le damos a continuación:

1. *Evitar sentencias PEEK y POKE,* ya que su uso depende en gran medida de la computadora específica en que se aplica. Si no puede evitar su uso, documéntelas suficientemente para que puedan adaptarse fácilmente a otras computadoras.

2. *Asignación múltiples.* Algunos BASIC permiten sentencias:

   — de la forma:

```
10 LET A = B = 0
```

— que asignan 0 a las variables B y C. Evite las asignaciones múltiples y utilize en su lugar:

```
10 LET A = 0 : LET B = 0
```

3. *Sentencias MAT.* Las sentencias MAT que realizan operaciones sobre matrices están disponibles *sólo* en las versiones de BASIC científico como son: HP 85- 87, VAX de Digital, Textronix, ANSI, etc. Utilize en su lugar bucles FOR-NEXT y los métodos de tratamiento de matrices (tablas y listas) dados en el capítulo 8.
4. *Comentarios.* Añadir gran número de comentarios.
5. *Evitar el uso de palabras- clave* como nombres de variables; la mayoría de las versiones BASIC no las admiten como nombres válidos.
6. *El tratamiento de cadenas* difiere de unos BASIC a otros. LEFT$, RIGHT$ y MID$ no existen en todas las versiones.
7. *Espacios en blanco no significativos.* Existen versiones de BASIC que requieren espacios en blanco entre las palabras clave y las variables, otros no:

```
10 FOR I =1 TO 50                10 FORI=1 TO50
```

APENDICES

I

Código ASCII

El Código ASCII (siglas de American Code for Information Interchange) es el Código Normalizado para el Intercambio de Información. Es un código internacional utilizado por la mayoría de las computadoras (con la excepción de las grandes computadoras fabricadas por IBM que siguen su código particular EBCDIC; sin embargo la computadora IBM/PC utiliza el código ASCII) para representar informaciones alfanuméricas: cifras, letras, signos, etc.

El código puede ser de 7 o de 8 bits (128 o 256 combinaciones posibles). Las computadoras más usuales utilizan el código de 8 bits dejando el octavo bit fijo (0 o 1) si se trabaja con 7 bits, o bien variable si se trabaja con 8 bits.

El código ASCII es estándar y las informaciones alfanuméricas más importantes tienen siempre el mismo código; así, es usual que las letras mayúsculas se representen con los códigos 65 a 90 (A a Z); el código de Timbre o Alarma (BEL) es el 7; el código de Avance de Línea (LF) es 10, etc. Sin embargo, algunos fabricantes suelen cambiar algunos códigos menos importantes o que se utilizan como letras minúsculas, símbolos gráficos, movimientos de cursor, etc. Por ello le aconsejamos que consulte el Manual de Usuario de su computadora o bien realice un listado completo del Código ASCII tal como se vio en el Capítulo 5 mediante la función CHR$. En este Apéndice le mostramos los códigos ASCII de las computadoras que se analizan a lo largo del texto.

Tenga presente también que el Código ASCII de su computadora no tiene por qué coincidir con el de su impresora, por lo cual deberá hacer dos listados, uno para cada dispositivo. La razón radica en la forma diferente de representación de una impresora y de una pantalla.

Así, por ejemplo, el DRAGON 64 no permite visualizar letras minúsculas en pantalla y sí en impresora, cuando se pulsan previamente las teclas SHIFT y 0. En este caso, en pantalla, la información aparece en campo inverso, mientras que en la impresora la misma presentación aparece en letras minúsculas.

Normalmente los Códigos ASCII de 0 a 31 suelen ser funciones de control: 7 (BEL) pitido, "bip", timbre; 13 (CR) retorno de carro, mueve el cursor a la siguiente línea; 21 (TCB) espacio hacia delante, etc. Los códigos 32 a 127 suelen ser caracteres de texto: 65 (letra A), 72 (letra H), 106 (letra j), 119 (letra w), 61 (signo =), etc. Los códigos 128 a 255 suelen representar caracteres gráficos, colores, sonidos, palabras claves de BASIC, etc. Tal vez en estos códigos altos (128 a 225) sea donde más se diferencian unos fabricantes de otros, ya que se utilizan para representar aquellas diferencias más acusadas entre cada modelo. Sin embargo, insistimos en que el fabricante puede variar algunos códigos. Un ejemplo representativo de ello es Commodore que utiliza para representar símbolos gráficos los códigos 97 a 122, que normalmente se emplean para representar las letras minúsculas.

Código de caracteres ASCII

Decimal	*Carácter*	*Decimal*	*Carácter*	*Decimal*	*Carácter*
000	NUL	043	+	086	V
001	SCH	044	,	087	W
002	STX	045	—	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	^
009	HT	052	4	095	—
010	LF	053	5	096	
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESC	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SP	075	K	118	v
033	¡	076	L	119	w
034	”	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	
038	&	081	Q	124	
039		082	R	125	
040	(	083	S	126	
041	)	084	T	127	DEL
042	*	085	U		

Códigos alfabéticos o de control ASCII

Decimal	*Código*	*Significado en inglés*	*Significado en español*
00	NUL	Null	Carácter nulo
01	SOH	Start of Heading	Principio de cabecera
02	STX	Start of Text	Principio de texto
03	ETX	End of Text	Fin de texto
04	EOT	End of Transmission	Fin de Transmisión
05	ENQ	Enquiry	Interrogación, consulta
06	ACK	Acknowledge	Acuse de recibo
07	BEL	Bell (audible signal)	Timbre o alarma
08	BS	Backspace	Retroceso
09	HT	Horizontal Tabulation	Tabulación horizontal
10	LF	Line Feed	Avance de una línea
11	VT	Vertical Tabulation	Tabulación vertical
12	FF	Form Feed	Avance de página
13	CR	Carriage Return	Retorno de carro
14	SO	Shift Out	Desplazamiento de salida
15	SI	Shift In	Desplazamiento de entrada
16	DLE	Data Link Escape	Cambio línea de datos
17	DC1	Device control 1	Dispositivo de control 1
18	DC2	Device control 2	Dispositivo de control 2
19	DC3	Device control 3	Dispositivo de control 3
20	DC4	Device control 4	Dispositivo de control 4
21	NAK	Negative Acknowledge	Acuse de no recepción
22	SYN	Synchronous Idle	Sincronización
23	ETB	End of transmission Block	Fin de bloque de transmisión
24	CAN	Cancel	Cancelación
25	EM	End of Medium	Fin del dispositivo
26	SUB	Substitute	Sustitución
27	ESC	Escale	Cambio, escape
28	FS	File Separator	Separador de fichero
29	GS	Group Separator	Separador de grupo
30	RS	Record Sparator	Separador de registro
31	US	Unit Separator	Separador de unidad
32	SP	Space	Espacio, barra espaciadora
127	DEL	Delete	Eliminar, suprimir

Código ASCII IBM/PC

Valor decimal	*Valor hexadecimal*	*Carácter de control*	*Carácter*
0	00	NUL	Nulo
1	01	SOH	☺
2	02	STX	☻
3	03	ETX	♥
4	04	EOT	♦
5	05	ENQ	♣
6	06	ACK	♠
7	07	BEL	Zumbido, Pitido, "bip"
8	08	BS	◘
9	09	HT	Tabulación
10	0A	LF	Avance de línea
11	0B	VT	Cursor a posición inicial
12	0C	FF	Avance de página
13	0D	CR	Retorno de carro, introducir
14	0E	SO	♫
15	0F	SI	☼
16	10	DLE	►
17	11	DC1	◄
18	12	DC2	↕
19	13	DC3	!!
20	14	DC4	¶
21	15	NAK	§
22	16	SYN	▬
23	17	ETB	↨
24	18	CAN	↑
25	19	EM	↓
26	1A	SUB	→
27	1B	ESC	←
28	IC	FS	Cursor a derecha
29	1D	GS	Cursor a izquierda
30	1E	RS	Cursor arriba
31	1F	US	Cursor abajo
32	20		Espacio
33	21		!
34	22		”
35	23		#
36	24		$
37	25		%
38	26		&
39	27		’
40	28		(
41	29		)
42	2A		*
43	2B		+
44	2C		,
45	2D		—
46	2E		.
47	2F		/
48	30		0
49	31		1
50	32		2
51	33		3
52	34		4
53	35		5
54	36		6
55	37		7
56	38		8

Código ASCII IBM/PC. *(Continuación)*

Valor decimal	*Valor hexadecimal*	*Carácter de control*	*Carácter*
57	39		9
58	3A		:
59	3B		;
60	3C		<
61	3D		=
62	3E		>
63	3F		?
64	40		@
65	41		A
66	42		B
67	43		C
68	44		D
69	45		E
70	46		F
71	47		G
72	48		H
73	49		I
74	4A		J
75	4B		K
76	4C		L
77	4D		M
78	4E		N
79	4F		O
80	50		P
81	51		Q
82	52		R
83	53		S
84	54		T
85	55		U
86	56		V
87	57		W
88	58		X
89	59		Y
90	5A		Z
91	5B		[
92	5C		\
93	5D		]
94	5E		^
95	5F		_
96	60		‘
97	61		a
98	62		b
99	63		c
100	64		d
101	65		e
102	66		f
103	67		g
104	68		h
105	69		i
106	6A		j
107	6B		k
108	6C		l
109	6D		m
110	6E		n
111	6F		o
112	70		p
113	71		q

Código ASCII IBM/PC. *(Continuación)*

Valor decimal	*Valor hexadecimal*	*Carácter de control*	*Carácter*
114	72		r
115	73		s
116	74		t
117	75		u
118	76		v
119	77		w
120	78		x
121	79		y
122	7A		z
123	7B		{
124	7C		¦
125	7D		}
126	7E		~
127	7F		⌂
128	80		Ç
129	81		ü
130	82		é
131	83		a
132	84		ä
133	85		à
134	86		a
135	87		ç
136	88		e
137	89		ë
138	8A		è
139	8B		ï
140	8C		ı
141	8D		ì
142	8E		Ä
143	8F		Á
144	90		É
145	91		ae
146	92		AE
147	93		o
148	94		ö
149	95		ò
150	96		u
151	97		ù
152	98		ÿ
153	99		Ö
154	9A		Ü
155	9B		¢
156	9C		£
157	9D		¥
158	9E		Pt
159	9F		*f*
160	A0		á
161	A1		í
162	A2		ó
163	A3		ú
164	A4		ñ
165	A5		Ñ
166	A6		*a*
167	A7		*o*
168	A8		¿
169	A9		⌐
170	AA		¬

Código ASCII IBM/PC. *(Continuación)*

Valor decimal	*Valor hexadecimal*	*Carácter de control*	*Carácter*
171	AB		½
172	AC		¼
173	AD		¡
174	AE		«
175	AF		»
176	B0		░
177	B1		▒
178	B2		▓
179	B3		│
180	B4		┤
181	B5		╡
182	B6		╢
183	B7		╖
184	B8		╕
185	B9		╣
186	BA		║
187	BB		╗
188	BC		╝
189	BD		╜
190	BE		╛
191	BF		┐
192	C0		└
193	C1		┴
194	C2		┬
195	C3		├
196	C4		─
197	C5		┼
198	C6		╞
199	C7		╟
200	C8		╚
201	C9		╔
202	CA		╩
203	CB		╦
204	CC		╠
205	CD		═
206	CE		╬
207	CF		╧
208	D0		╨
209	D1		╤
210	D2		╥
211	D3		╙
212	D4		╘
213	D5		╒
214	D6		╓
215	D7		╫
216	D8		╪
217	D9		┘
218	DA		┌
219	DB		█
220	DC		▄
221	DD		▌
222	DE		▐
223	DF		▀
224	E0		α
225	E1		β
226	E2		Γ
227	E3		π

Código ASCII IBM/PC. ***(Continuación)***

Valor de lineal	*Valor hexadecimal*	*Carácter de control*	*Carácter*
228	E4		Σ
229	E5		σ
230	E6		μ
231	E7		τ
232	E8		Φ
233	E9		θ
234	EA		Ω
235	EB		δ
236	EC		∞
237	ED		Ø
238	EE		∈
239	EF		∩
240	F0		≡
241	F1		±
242	F2		≥
243	F3		≤
244	F4		⌠
245	F5		⌡
246	F6		÷
247	F7		≈
248	F8		°
249	F9		•
250	FA		·
251	FB		√
252	FC		ⁿ
253	FD		²
254	FE		■
255	FF		(espacio en blanco 'FF')

APPLE II

Código de caracteres ASCII y valores simbólicos de palabras reservadas Applesoft

La primera tabla de este apéndice muestra los códigos de 0 a 127 y los caracteres que representan. Los códigos ASCII en el intervalo de 128 a 255 repiten los mismos caracteres en la pantalla de visualización del Apple II que los códigos 0 a 127.

La segunda tabla de este apéndice lista las palabras reservadas Applesoft. Cada palabra reservada se representa por un código, denominado valor simbólico («token»), en el rango de 128 a 255 y ocupa un byte en memoria. El valor simbólico sustituye al deletreo de palabras reservadas en la memoria del Apple II y el disco. Esta lista se proporciona en orden numérico de valores simbólicos.

La tabla muestra la relación entre los códigos ASCII y los códigos utilizados para representar caracteres en la pantalla. Los códigos de pantalla también representan información sobre tipos de caracteres normal, inverso y parpadeante.

Caracteres, pulsaciones y códigos ASCII. Apple II

Código ASCII	*Visualización carácter en pantalla*		*Pulsaciones de teclas*	
	Apple IIe	*Otros modelos*	*Apple IIe*	*Otros modelos*
0	Ninguno	Ninguno	CTRL-@	CTRL-@
1	Ninguno	Ninguno	CTRL-A	CTRL-A
2	Ninguno	Ninguno	CTRL-B	CTRL-B
3	Ninguno	Ninguno	CTRL-C	CTRL-C
4	Ninguno	Ninguno	CTRL-D	CTRL-D
5	Ninguno	Ninguno	CTRL-E	CTRL-E
6	Ninguno	Ninguno	CTRL-F	CTRL-F
7	Pitido	Pitido	CTRL-G	CTRL-G
8	Retroceso espacio	Retroceso espacio	CTRL-H	CTRL-H
9	Ninguno	Ninguno	CTRL-I	CTRL-I
10	Avance de línea	Avance de línea	CTRL-J	CTRL-J
11	Ninguno	Ninguno	CTRL-K	CTRL-K
12	Ninguno	Ninguno	CTRL-L	CTRL-L
13	Return	Return	CTRL-M	CTRL-M
14	Ninguno	Ninguno	CTRL-N	CTRL-N
15	Ninguno	Ninguno	CTRL-O	CTRL-O
16	Ninguno	Ninguno	CTRL-P	CTRL-P
17	Ninguno	Ninguno	CTRL-Q	CTRL-Q
18	Ninguno	Ninguno	CTRL-R	CTRL-R
19	Ninguno	Ninguno	CTRL-S	CTRL-S
20	Ninguno	Ninguno	CTRL-T	CTRL-T
21	Ninguno	Espacio adelante	CTRL-U	CTRL-U
22	Ninguno	Ninguno	CTRL-V	CTRL-V
23	Ninguno	Ninguno	CTRL-W	CTRL-W
24	Ninguno	Cancelar línea	CTRL-X	CTRL-X
25	Ninguno	Ninguno	CTRL-Y	CTRL-Y
26	Ninguno	Ninguno	CTRL-Z	CTRL-Z
27	Ninguno	Ninguno	ESC	ESC
28	Ninguno	Ninguno	CTRL-\	Ninguno
29	Ninguno	Ninguno	CTRL-]	CTRL-SHIFT-M
30	Ninguno	Ninguno	Ninguno	CTRL-^
31	Ninguno	Ninguno	CTRL-SHIFT--	Ninguno
32	Espacio	Espacio	BARRA ESPACIADORA	BARRA ESPACIADORA
33	!	!	SHIFT-1	SHIFT-1
34	”	”	SHIFT-’	SHIFT-2
35	#	#	SHIFT-3	SHIFT-3
36	$	$	SHIFT-4	SHIFT-4
37	%	%	SHIFT-5	SHIFT-5
38	&	&	SHIFT-7	SHIFT-6
39	'	'		SHIFT-7
40	(	(	SHIFT-9	SHIFT-8
41	)	)	SHIFT-0	SHIFT-9
42	*	*	SHIFT-8	SHIFT-:
43	+	+	SHIFT-=	SHIFT-;
44	,	,	,	,
45	-	-	-	-
46	.	.	.	.
47	/	/	/	/
48	0	0	0	0
49	1	1	1	1

Caracteres, pulsaciones y códigos ASCII. Apple II. ***(Continuación)***

Código ASCII	*Visualización carácter en pantalla*		*Pulsaciones de teclas*	
	Apple IIe	*Otros modelos*	*Apple IIe*	*Otros modelos*
50	2	2	2	2
51	3	3	3	3
52	4	4	4	4
53	5	5	5	5
54	6	6	6	6
55	7	7	7	7
56	8	8	8	8
57	9	9	9	9
58	:	:	SHIFT-;	:
59	;	;	;	;
60	<	<	SHIFT-,	SHIFT-,
61	=	=	=	SHIFT--
62	>	>	SHIFT-.	SHIFT-.
63	?	?	SHIFT-/	SHIFT-/
64	@	@	SHIFT-2	SHIFT-P
65	A	A	SHIFT-A	A
66	B	B	SHIFT-B	B
67	C	C	SHIFT-C	C
68	D	D	SHIFT-D	D
69	E	E	SHIFT-E	E
70	F	F	SHIFT-F	F
71	G	G	SHIFT-G	G
72	H	H	SHIFT-H	H
73	I	I	SHIFT-I	I
74	J	J	SHIFT-J	J
75	K	K	SHIFT-K	K
76	L	L	SHIFT-L	L
77	M	M	SHIFT-M	M
78	N	N	SHIFT-N	N
79	O	O	SHIFT-O	O
80	P	P	SHIFT-P	P
81	Q	Q	SHIFT-Q	Q
82	R	R	SHIFT-R	R
83	S	S	SHIFT-S	S
84	T	T	SHIFT-T	T
85	U	U	SHIFT-U	U
86	V	V	SHIFT-V	V
87	W	W	SHIFT-W	W
88	X	X	SHIFT-X	X
89	Y	Y	SHIFT-Y	Y
90	Z	Z	SHIFT-Z	Z
91	[	[	[	Ninguno

Caracteres, pulsaciones y códigos ASCII. Apple II. *(Continuación)*

<table>
<tr><th rowspan="2">Código ASCII</th><th colspan="2">Visualización carácter en pantalla</th><th colspan="2">Pulsaciones de teclas</th></tr>
<tr><th>Apple IIe</th><th>Otros modelos</th><th>Apple IIe</th><th>Otros modelos</th></tr>
<tr><td>92</td><td>\</td><td>\</td><td>\</td><td>Ninguno</td></tr>
<tr><td>93</td><td>]</td><td>]</td><td>]</td><td>SHIFT-M</td></tr>
<tr><td>94</td><td>^</td><td>^</td><td>SHIFT-6</td><td>SHIFT-N</td></tr>
<tr><td>95</td><td>_</td><td>Ninguno</td><td>SHIFT--</td><td>Ninguno</td></tr>
<tr><td>96</td><td>`</td><td>Ninguno</td><td>`</td><td>Ninguno</td></tr>
<tr><td>97</td><td>a</td><td>A</td><td>A</td><td>Ninguno</td></tr>
<tr><td>98</td><td>b</td><td>B</td><td>B</td><td>Ninguno</td></tr>
<tr><td>99</td><td>c</td><td>C</td><td>C</td><td>Ninguno</td></tr>
<tr><td>100</td><td>d</td><td>D</td><td>D</td><td>Ninguno</td></tr>
<tr><td>101</td><td>e</td><td>E</td><td>E</td><td>Ninguno</td></tr>
<tr><td>102</td><td>f</td><td>F</td><td>F</td><td>Ninguno</td></tr>
<tr><td>103</td><td>g</td><td>G</td><td>G</td><td>Ninguno</td></tr>
<tr><td>104</td><td>h</td><td>H</td><td>H</td><td>Ninguno</td></tr>
<tr><td>105</td><td>i</td><td>I</td><td>I</td><td>Ninguno</td></tr>
<tr><td>106</td><td>j</td><td>J</td><td>J</td><td>Ninguno</td></tr>
<tr><td>107</td><td>k</td><td>K</td><td>K</td><td>Ninguno</td></tr>
<tr><td>108</td><td>l</td><td>L</td><td>L</td><td>Ninguno</td></tr>
<tr><td>109</td><td>m</td><td>M</td><td>M</td><td>Ninguno</td></tr>
<tr><td>110</td><td>n</td><td>N</td><td>N</td><td>Ninguno</td></tr>
<tr><td>111</td><td>o</td><td>O</td><td>O</td><td>Ninguno</td></tr>
<tr><td>112</td><td>p</td><td>P</td><td>P</td><td>Ninguno</td></tr>
<tr><td>113</td><td>q</td><td>Q</td><td>Q</td><td>Ninguno</td></tr>
<tr><td>114</td><td>r</td><td>R</td><td>R</td><td>Ninguno</td></tr>
<tr><td>115</td><td>s</td><td>S</td><td>S</td><td>Ninguno</td></tr>
<tr><td>116</td><td>t</td><td>T</td><td>T</td><td>Ninguno</td></tr>
<tr><td>117</td><td>u</td><td>U</td><td>U</td><td>Ninguno</td></tr>
<tr><td>118</td><td>v</td><td>V</td><td>V</td><td>Ninguno</td></tr>
<tr><td>119</td><td>w</td><td>W</td><td>W</td><td>Ninguno</td></tr>
<tr><td>120</td><td>x</td><td>X</td><td>X</td><td>Ninguno</td></tr>
<tr><td>121</td><td>y</td><td>Y</td><td>Y</td><td>Ninguno</td></tr>
<tr><td>122</td><td>z</td><td>Z</td><td>Z</td><td>Ninguno</td></tr>
<tr><td>123</td><td>{</td><td>Ninguno</td><td>SHIFT-[</td><td>Ninguno</td></tr>
<tr><td>124</td><td>|</td><td>Ninguno</td><td>SHIFT-\</td><td>Ninguno</td></tr>
<tr><td>125</td><td>}</td><td>Ninguno</td><td>SHIFT-]</td><td>Ninguno</td></tr>
<tr><td>126</td><td>~</td><td>Ninguno</td><td>SHIFT-`</td><td>Ninguno</td></tr>
<tr><td>127</td><td>▒</td><td>Ninguno</td><td>DELETE</td><td>Ninguno</td></tr>
</table>

Valores simbólicos (tokens) de las palabras reservadas de Applesoft

Palabra reservada	*Token*	*Palabra reservada*	*Token*	*Palabra reservada*	*Token*
ABS	(212)	HTAB	(150)	REM	(178)
ABD	(205)	IF	(173)	RESTORE	(174)
ASC	(230)	IN#	(139)	RESUME	(166)
AT	(197)	INPUT	(132)	RETURN	(177)
ATN	(225)	INT	(211)	RIGHT$	(233)
CALL	(140)	INVERSE	(158)	RND	(219)
CHR$	(231)	LEFT$	(232)	ROT=	(152)
CLEAR	(189)	LEN	(227)	RUN	(172)
COLOR=	(160)	LET	(170)	SAVE	(183)
CONT	(187)	LIST	(188)	SCALE=	(153)
COS	(222)	LOAD	(182)	SCRN(	(215)
DATA	(131)	LOG	(220)	SGN	(210)
DEF	(184)	LOMEM:	(164)	SHLOAD	(154)
DEL	(133)	MID$	(234)	SIN	(223)
DIM	(134)	NEW	(191)	SPC(	(195)
END	(128)	NEXT	(130)	SPEED=	(169)
EXP	(221)	NORMAL	(157)	SQR	(218)
FLASH	(159)	NOT	(198)	STEP	(199)
FN	(194)	NOTRACE	(156)	STOP	(179)
FOR	(129)	ON	(180)	STORE	(168)
FRE	(214)	ONERR	(165)	STR$	(228)
GET	(190)	OR	(206)	TAB(	(192)
GOSUB	(176)	PDL	(216)	TAN	(224)
GOTO	(171)	PEEK	(226)	TEXT	(137)
GR	(136)	PLOT	(141)	THEN	(196)
HCOLOR=	(146)	POKE	(185)	TO	(193)
HGR	(145)	POP	(161)	TRACE	(155)
HGR2	(144)	POS	(217)	USR	(213)
HIMEM:	(163)	PRINT	(186)	VAL	(229)
HLIN	(142)	PR#	(138)	VLIN	(143)
HOME	(151)	READ	(135)	VTAB	(162)
HPLOT	(147)	RECALL	(167)	WAIT	(181)
				XDRAW	(149)

Código ASCII. HP-85, 86, 87

FORMAS EQUIVALENTES				*FORMAS EQUIVALENTES*				*FORMAS EQUIVALENTES*				*FORMAS EQUIVALENTES*			
Car.	*Binario*	*Octal*	*Dec.*	*Car.*	*Binario*	*Octal*	*Dec.*	*Car.*	*Binario*	*Octal*	*Dec.*	*Car.*	*Binario*	*Octal*	*Dec.*
◄ @^{c}	00000000	000	0	SPACE	00100000	040	32	⊵	01000000	100	64	\ [KEY LABEL]^{s}	01100000	140	96
¿ A^{c}	00000001	001	1	!	00100001	041	33	A	01000001	101	65	a	01100001	141	97
X̄ B^{c}	00000010	002	2	"	00100010	042	34	B	01000010	102	66	b	01100010	142	98
N̄ C^{c}	00000011	003	3	#	00100011	043	35	C	01000011	103	67	c	01100011	143	99
α D^{c}	00000100	004	4	$	00100100	044	36	D	01000100	104	68	d	01100100	144	100
β E^{c}	00000101	005	5	%	00100101	045	37	E	01000101	105	69	e	01100101	145	101
Γ F^{c}	00000110	006	6	&	00100110	046	38	F	01000110	106	70	f	01100110	146	102
n̄ G^{c}	00000111	007	7	'	00100111	047	39	G	01000111	107	71	g	01100111	147	103
Δ H^{c}	00001000	010	8	(	00101000	050	40	H	01001000	110	72	h	01101000	150	104
σ I^{c}	00001001	011	9	)	00101001	051	41	I	01001001	111	73	i	01101001	151	105
↑ J^{c}	00001010	012	10	*	00101010	052	42	J	01001010	112	74	j	01101010	152	106
⋱ K^{c}	00001011	013	11	+	00101011	053	43	K	01001011	113	75	k	01101011	153	107
μ L^{c}	00001100	014	12	,	00101100	054	44	L	01001100	114	76	l	01101100	154	108
M^{c}	00001101	015	13	–	00101101	055	45	M	01001101	115	77	m	01101101	155	109
τ N^{c}	00001110	016	14	.	00101110	056	46	N	01001110	116	78	n	01101110	156	110
Φ O^{c}	00001111	017	15	/	00101111	057	47	O	01001111	117	79	o	01101111	157	111
Θ P^{c}	00010000	020	16	0	00110000	060	48	P	01010000	120	80	p	01110000	160	112
Ω Q^{c}	00010001	021	17	1	00110001	061	49	Q	01010001	121	81	q	01110001	161	113
δ R^{c}	00010010	022	18	2	00110010	062	50	R	01010010	122	82	r	01110010	162	114
H S^{c}	00010011	023	19	3	00110011	063	51	S	01010011	123	83	s	01110011	163	115
á T^{c}	00010100	024	20	4	00110100	064	52	T	01010100	124	84	t	01110100	164	116
H U^{c}	00010101	025	21	5	00110101	065	53	U	01010101	125	85	u	01110101	165	117
ä V^{c}	00010110	026	22	6	00110110	066	54	V	01010110	126	86	v	01110110	166	118
u W^{c}	00010111	027	23	7	00110111	067	55	W	01010111	127	87	w	01110111	167	119
ö X^{c}	00011000	030	24	8	00111000	070	56	X	01011000	130	88	x	01111000	170	120
Ü Y^{c}	00011001	031	25	9	00111001	071	57	Y	01011001	131	89	y	01111001	171	121
ü Z^{c}	00011010	032	26	:	00111010	072	58	Z	01011010	132	90	z	01111010	172	122
Ɇ [^{c}	00011011	033	27	;	00111011	073	59	[	01011011	133	91	n [/]^{s}	01111011	173	123
œ \\$^{c}$	00011100	034	28	<	00111100	074	60	\	01011100	134	92		01111100	174	124
2]^{c}	00011101	035	29	=	00111101	075	61	]	01011101	135	93	→ [−]^{s}	01111101	175	125
£ ∧^{c}	00011110	036	30	>	00111110	076	62	ˆ	01011110	136	94	Σ [•]^{s}	01111110	176	126
░ –^{c}	00011111	037	31	?	00111111	077	63	–	01011111	137	95	⊢ [+]^{s}	01111111	177	127

1. *Acceso a caracteres 0 a 31 con la pulsación combinada de las teclas CTRL + carácter.*
2. *Acceso a caracteres 173 a 177 con la pulsación combinada de las teclas SHIFT + carácter.*

Código ASCII. Amstrad (CPC 464, CPC 664, CPC 6128)

DEC	OCTAL	HEX	Caracteres ASCII	DEC	OCTAL	HEX	ASCII	DEC	OCTAL	HEX	ASCII
0	000	00	NUL ([CONTROL]@)	50	062	32	2	100	144	64	d
1	001	01	SOH ([CONTROL]A)	51	063	33	3	101	145	65	e
2	002	02	STX ([CONTROL]B)	52	064	34	4	102	146	66	f
3	003	03	ETX ([CONTROL]C)	53	065	35	5	103	147	67	g
4	004	04	EOT ([CONTROL]D)	54	066	36	6	104	150	68	h
5	005	05	ENQ ([CONTROL]E)	55	067	37	7	105	151	69	i
6	006	06	ACK ([CONTROL]F)	56	070	38	8	106	152	6A	j
7	007	07	BRL ([CONTROL]G)	57	071	39	9	107	153	6B	k
8	010	08	BS ([CONTROL]H)	58	072	3A	:	108	154	6C	l
9	011	09	HT ([CONTROL]I)	59	073	3B	;	109	155	6D	m
10	012	0A	LF ([CONTROL]J)	60	074	3C	<	110	156	6E	n
11	013	0B	VT ([CONTROL]K)	61	075	3D	=	111	157	6F	o
12	014	0C	FF ([CONTROL]L)	62	076	3E	>	112	160	70	p
13	015	0D	CR ([CONTROL]M)	63	077	3F	?	113	161	71	q
14	016	0E	SO ([CONTROL]N)	64	100	40	@	114	162	72	r
15	017	0F	SI ([CONTROL]O)	65	101	41	A	115	163	73	s
16	020	10	DLE ([CONTROL]P)	66	102	42	B	116	164	74	t
17	021	11	DC1 ([CONTROL]Q)	67	103	43	C	117	165	75	u
18	022	12	DC2 ([CONTROL]R)	68	104	44	D	118	166	76	v
19	023	13	DC3 ([CONTROL]S)	69	105	45	E	119	167	77	w
20	024	14	DC4 ([CONTROL]T)	70	106	46	F	120	170	78	x
21	025	15	NAK ([CONTROL]U)	71	107	47	G	121	171	79	y
22	026	16	SYN ([CONTROL]V)	72	110	48	H	122	172	7A	z
23	027	17	ETB ([CONTROL]W)	73	111	49	I	123	173	7B	{
24	030	18	CAN ([CONTROL]X)	74	112	4A	J	124	174	7C	
25	031	19	EM ([CONTROL]Y)	75	113	4B	K	125	175	7D	}
26	032	1A	SUB ([CONTROL]Z)	76	114	4C	L	126	176	7E	~
27	033	1B	ESC ([CONTROL][)	77	115	4D	M				
28	034	1C	FS ([CONTROL]\)	78	116	4E	N				
29	035	1D	GS ([CONTROL]])	79	117	4F	O				
30	036	1E	RS ([CONTROL]↑)	80	120	50	P				
31	037	1F	US ([CONTROL]0)	81	121	51	Q				
32	040	20	SP	82	122	52	R				
33	041	21	!	83	123	53	S				
34	042	22	”	84	124	54	T				
35	043	23	#	85	125	55	U				
36	044	24	$	86	126	56	V				
37	045	25	%	87	127	57	W				
38	046	26	&	88	130	58	X				
39	047	27	´	89	131	59	Y				
40	050	28	(	90	132	5A	Z				
41	051	29	)	91	133	5B	[				
42	052	2A	*	92	134	5C	\				
43	053	2B	+	93	135	5D	]				
44	054	2C	,	94	136	5E	↑				
45	055	2D	–	95	137	5F	_				
46	056	2E	.	96	140	60	`				
47	057	2F	/	97	141	61	a				
48	060	30	0	98	142	62	b				
49	061	31	1	99	143	63	c				

Código ASCII. Computadoras MSX.

Número	*Carácter*	*Número*	*Carácter*	*Número*	*Carácter*	*Número*	*Carácter*
1	☺	27	┘	53	5	79	O
2	☻	28	╳	54	6	80	P
3	♥	29	╱	55	7	81	Q
4	♦	30	╲	56	8	82	R
5	♣	31	┼	57	9	83	S
6	♠	32		58	:	84	T
7	•	33	!	59	;	85	U
8	◘	34	”	60	<	86	V
9	○	35	#	61	=	87	W
10	◙	36	$	62	>	88	X
11	♂	37	%	63	?	89	Y
12	♀	38	&	64	@	90	Z
13	♪	39	’	65	A	91	[
14	♫	40	(	66	B	92	\
15	☼	41	)	67	C	93	]
16	┼	42	*	68	D	94	^
17	┴	43	+	69	E	95	–
18	┬	44	’	70	F	96	
19	┤	45	E̲	71	G	97	a
20	├	46	.	72	H	98	b
21	┼	47		73	I	99	c
22	│	48	0	74	J	100	d
23	─	49	1	75	K	101	e
24	┌	50	2	76	L	102	f
25	┐	51	3	77	M	103	g
26	└	52	4	78	N	104	h

Código ASCII. Computadoras MSX. ***(Continuación)***

Número	*Carácter*	*Número*	*Carácter*	*Número*	*Carácter*	*Número*	*Carácter*
105	i	131	â	157	Y	183	ũ
106	j	132	ä	158	Pt	184	ij
107	k	133	à	159	*f*	185	ij
108	l	134	å	160	á	186	3/4
109	m	135	ç	161	í	187	~
110	n	136	ê	162	ó	188	◇
111	o	137	ë	163	ú	189	‰
112	p	138	è	164	ñ	190	¶
113	q	139	ï	165	Ñ	191	§
114	r	140	î	166	a̲	192	▬
115	s	141	ì	167	o̲	193	▜
116	t	142	Ä	168	¿	194	■
117	u	143	Å	169	⌐	195	▬
118	v	144	É	170	¬	196	▬
119	w	145	æ	171	½	197	▮
120	x	146	Æ	172	¼	198	▌
121	y	147	ô	173	¡		▟
122	z	148	ö	174	<<	200	█
123	{	149	ò	175	>>	201	▐
124	|	150	û	176	Ã	202	█
125	}	151	ù	177	ã	203	///
126	~	152	ÿ	178	Ĩ	204	\\\
127		153	Ö	179	ĩ	205	▼
128	Ç	154	Ü	180	Õ	206	▲
129	ü	155	ç	181	õ	207	▶
130	é	156	£	182	Ũ	208	◀

Código ASCII. Computadoras MSX. ***(Continuación)***

Número	*Carácter*	*Número*	*Carácter*	*Número*	*Carácter*	*Número*	*Carácter*
209	⧗	221	▌	233	θ	245	⌡
210	⋈	222	▐	234	Ω	246	÷
211	▪	223	▀	235	δ	247	≈
212	▮	224	α	236	∞	248	○
213	▮	225	β	237		250	●
214	▮	226	Γ	238	ε	250	·
215	▓	227	Π	239	η	251	√
216	^	228	Σ	240	≡	252	^{n}
217	↕	229	σ	241	±	253	^{2}
218	ω	230	μ	242	≧	254	▮
219	█	231	γ	243	≦	255	
220	▬	232	Φ	244	⌠		

Código ASCII Commodore 64 y Commodore 128

Imprime	CHR$	Imprime	CHR$	Imprime	CHR$	Imprime	CHR$
	0	CLR HOME	19	&	38	:	58
	1	INST DEL	20	'	39	;	59
	2		21	(	40	<	60
	3		22	)	41	=	61
	4		23	*	42	>	62
WHT	5		24	+	43	?	63
	6		25	,	44	@	64
	7		26	-	45	A	65
	8		27	.	46	B	66
	9	RED	28	/	47	C	67
	10	CRSR →	29	0	48	D	68
	11	GRN	30	1	49	E	69
	12	BLU	31	2	50	F	70
RETURN	13		32	3	51	G	71
CONMUTACION A MINUSCULAS	14	!	33	4	52	H	72
	15	"	34	5	53	I	73
	16	#	35	6	54	J	74
CRSR ↓	17	$	36	7	55	K	75
RVS ON	18	%	37	8	56	L	76
				9	57	M	77

Código ASCII Commodore 64 y Commodore 128. *(Continuación)*

Imprime	CHR$	Imprime	CHR$	Imprime	CHR$	Imprime	CHR$
N	78		98		118	f4	138
O	79		99		119	f6	139
P	80		100		120	f8	140
Q	81		101		121	SHIFT RETURN	141
R	82		102		122	CONMU-TACION A MAYUS-CULAS	142
S	83		103		123		143
T	84		104		124	BLK	144
U	85		105		125	↑ CRSR	145
V	86		106		126	RVS OFF	146
W	87		107		127	CLR HOME	147
X	88		108		128	INST DEL	148
Y	89		109		129		149
Z	90		110		130		150
[	91		111		131		151
£	92		112		132		152
]	93		113	f1	133		153
↑	94		114	f3	134		154
←	95		115	f5	135		
	96		116	f7	136		
	97		117	f2	137		

Código ASCII Commodore 64 y Commodore 128. *(Continuación)*

Imprime	CHR$	Imprime	CHR$	Imprime	CHR$	Imprime	CHR$
	155		175		195		215
PUR	156		176		196		216
CRSR ←	157		177		197		217
YES	158		178		198		218
CIAN ESPACIO	159		179		199		219
	160		180		200		220
	161		181		201		221
	162		182		202		222
	163		183		203		223
	164		184		204		224
	165		185		205		225
	166		186		206		226
	167		187		207		227
	168		188		208		228
	169		189		209		229
	170		190		210		230
	171		191		211		231
	172		192		212		232
	173		193		213		233
	174		194		214		234

Código ASCII Commodore 64 y Commodore 128. *(Continuación)*

Imprime	CHR$	Imprime	CHR$	Imprime	CHR$	Imprime	CHR$
	235		241		246		251
	236		242		247		252
	237		243		248		253
	238		244		249		254
	239		245		250		255
	240						

Código ASCII ZX Spectrum.

Código		*Carácter*	*Código*		*Carácter*	*Código*		*Carácter*
Dec.	**Hex.**		**Dec.**	**Hex.**		**Dec.**	**Hex.**	
Ø	ØØ		57	39	9	114	72	r
1	Ø1		58	3A	:	115	73	s
2	Ø2	no utilizados	59	3B	;	116	74	t
3	Ø3		6Ø	3C	<	117	75	u
4	Ø4		61	3D	=	118	76	v
5	Ø5		62	3E	>	119	77	w
6	Ø6	PRINT coma	63	3F	?	12Ø	78	x
7	Ø7	cursor a izquierda	64	4Ø	@	121	79	y
8	Ø8	cursor a derecha	65	41	A	122	7A	z
9	Ø9	cursor abajo	66	42	B	123	7B	{
1Ø	ØA	cursor arriba	67	43	C	124	7C	|
11	ØB	DELETE	68	44	D	125	7D	}
12	ØC	ENTER	69	45	E	126	7E	-
13	ØD	nombre	7Ø	46	F	127	7F	©
14	ØE	no utilizado	71	47	G	128	8Ø	□
15	ØF	control INK	72	48	H	129	81	▝
16	1Ø	control PAPER	73	49	I	13Ø	82	▘
17	11	control FLASH	74	4A	J	131	83	▀
18	12	control BRIGHT	75	4B	K	132	84	▗
19	13	control INVERSE	76	4C	L	133	85	▐
2Ø	14	control OVER	77	4D	M	134	86	▚
21	15	control AT	78	4E	N	135	87	▜
22	16	control TAB	79	4F	O	136	88	▖
23	17		8Ø	5Ø	P	137	89	▞
24	18		81	51	Q	138	8A	▌
25	19		82	52	R	139	8B	▛
26	1A		83	53	S	14Ø	8C	▄
27	1B		84	54	T	141	8D	▟
28	1C	no utilizados	85	55	U	142	8E	▙
29	1D		86	56	V	143	8F	█
3Ø	1E		87	57	W	144	9Ø	(a)
31	1F		88	58	X	145	91	(b)
32	2Ø	Espacio	89	59	Y	146	92	(c)
33	21	!	9Ø	5A	Z	147	93	(d)
34	22	"	91	5B	[	148	94	(e)
35	23	#	92	5C	/	149	95	(f)
36	24	$	93	5D	]	15Ø	96	(g)
37	25	%	94	5E	↑	151	97	(h)
38	26	&	95	5F	_	152	98	(i)
39	27	'	96	6Ø	£	153	99	(j) Caracteres gráficos usuario
4Ø	28	(	97	61	a	154	9A	(k)
41	29	)	98	62	b	155	9B	(l)
42	2A	*	99	63	c	156	9C	(m)
43	2B	+	1ØØ	64	d	157	9D	(n)
44	2C	,	1Ø1	65	e	158	9E	(o)
45	2D	—	1Ø2	66	f	159	9F	(p)
46	2E	.	1Ø3	67	g	16Ø	AØ	(q)
47	2F	/	1Ø4	68	h	161	A1	(r)
48	3Ø	Ø	1Ø5	69	i	162	A2	(s)
49	31	1	1Ø6	6A	j	163	A3	(t)
5Ø	32	2	1Ø7	6B	k	164	A4	(u)
51	33	3	1Ø8	6C	l	165	A5	**RND**
52	34	4	1Ø9	6D	m	166	A6	**INKEY$**
53	35	5	11Ø	6E	n	167	A7	**PI**
54	36	6	111	6F	o	168	A8	**FN**
55	37	7	112	7Ø	p	169	A9	**POINT**
56	38	8	113	71	q			

Código ASCII ZX Spectrum. ***(Continuación)***

Código		*Carácter*	*Código*		*Carácter*	*Código*		*Carácter*
Dec.	**Hex.**		**Dec.**	**Hex.**		**Dec.**	**Hex.**	
170	AA	SCREEN$	199	C7	<=	228	E4	DATA
171	AB	ATTR	200	C8	>=	229	E5	RESTORE
172	AC	AT	201	C9	<>	230	E6	NEW
173	AD	TAB	202	CA	LINE	231	E7	BORDER
174	AE	VAL$	203	CB	THEN	232	E8	CONTINUE
175	AF	CODE	204	CC	TO	233	E9	DIM
176	B0	VAL	205	CD	STEP	234	EA	REM
177	B1	LEN	206	CE	DEF FN	235	EB	FOR
178	B2	SIN	207	CF	CAT	236	EC	GO TO
179	B3	COS	208	D0	FORMAT	237	ED	GO SUB
180	B4	TAN	209	D1	MOVE	238	EE	INPUT
181	B5	ASN	210	D2	ERASE	239	EF	LOAD
182	B6	ACS	211	D3	OPEN #	240	F0	LIST
183	B7	ATN	212	D4	CLOSE #	241	F1	LET
184	B8	LN	213	D5	MERGE	242	F2	PAUSE
185	B9	EXP	214	D6	VERIFY	243	F3	NEXT
186	BA	INT	215	D7	BEEP	244	F4	POKE
187	BB	SQR	216	D8	CIRCLE	245	F5	PRINT
188	BC	SGN	217	D9	INK	246	F6	PLOT
189	BD	ABS	218	DA	PAPER	247	F7	RUN
190	BE	PEEK	219	DB	FLASH	248	F8	SAVE
191	BF	IN	220	DC	BRIGHT	249	F9	RANDOMIZE
192	C0	USR	221	DD	INVERSE	250	FA	IF
193	C1	STR$	222	DE	OVER	251	FB	CLS
194	C2	CHR$	223	DF	OUT	252	FC	DRAW
195	C3	NOT	224	E0	LPRINT	253	FD	CLEAR
196	C4	BIN	225	E1	LLIST	254	FE	RETURN
197	C5	OR	226	E2	STOP	255	FF	COPY
198	C6	AND	227	E3	READ			

II

Utilización de impresoras

Las impresoras actualmente más actualizadas tienen dos anchuras de carro normalizadas, 80 o 132, columnas que permiten representar en papel 80 o 132 caracteres por línea. Sin embargo, según el tipo de escritura, se podrá aumentar o disminuir el número de caracteres por línea.

El estudio de las características técnicas más sobresalientes y de los códigos de control más utilizados en la escritura usual con computadoras los centraremos en la marca de impresora EPSON que constituyen un estándar en el campo de las impresoras, seguido por numerosas marcas y modelos.

IMPRESORAS EPSON

Modelos FX-85/FX-105

Características estándar

— Método de impresión: impacto por matriz de puntos.

— Velocidad de impresión: 160 CPS.

— Espaciado de líneas: 1/6", 1/8" o programable.

— Anchura de carro: 80 en FX-85 (caracteres normales)
136 en FX-105 (caracteres normales)

Características de impresion

Juego de caracteres: 96 caracteres ASCII
96 caracteres cursiva
32 caracteres internacionales
32 caracteres gráficos
96 caracteres de letra de alta calidad
32 caracteres internacionales NLQ

Tipos de letras: Proporcional
Expandido
Cursiva
Comprimido
Subrayado
Doble pasada
Superíndice / Subíndice
Negrita
Doble ancho

Matriz de puntos: 9 × 9 puntos
18 × 18 puntos en NLQ

Códigos de caracteres de control

Tabla II.1. Caracteres de control.

Acción	Secuencia	Comentarios
Impresión ampliada	CHR$(14)	Se cancela automáticamente al final de la línea correspondiente
Retorno de carro	CHR$(13)	
Timbre (Pitido)	CHR$(7)	Suena el altavoz durante unos instantes.
Avance de línea	CHR$(10)	
Espaciado de línea 1/6”	CHR$(27)“2”	Espaciado por omisión. Códigos decimales 27, 50
Espaciado de línea 1/8”	CHR$(27)“0”	Códigos decimales 27, 48
Avance de página	CHR$(12)	
Establecer número de líneas/página	CHR$(27)“C”R	
Impresión comprimida	CHR$(15)	
Cancelar impresión comprimida	CHR$(18)	
Impresión enfatizada (negrita)	CHR$(27)“E”	Códigos decimales 27, 69
Cancelar impresión enfatizada	CHR$(27)“F”	Códigos decimales 27, 70
Doble pasada o impresión	CHR$(27)“G”	Códigos decimales 27, 71
Cancelar doble impresión	CHR$(27)“H”	Códigos decimales 27, 72
Superíndice	CHR$(27)“S”0	Códigos decimales 27, 83
Subíndice	CHR$(27)“S”1	Códigos decimales 27, 83
Cancelar superíndice/subíndice	CHR$(27)“T”	Códigos decimales 27, 84

Prácticas de impresión (Control de la impresora desde BASIC)

Este párrafo explica los métodos utilizados para obtener los caracteres especiales de la impresora utilizando el lenguaje de programación BASIC. Los códigos de control y los programas prácticos de demostración se han realizado con impresoras EPSON e IBM.

En la mayoría de las versiones de BASIC y, en particular en Microsoft BASIC (MBASIC, BASICA, GW-BASIC, MSX, BASIC-80, etc.), el método normal de producir salidas impresas es utilizar la sentencia/orden LPRINT.

Los *códigos de control* estándar de las impresoras EPSON se pueden enviar utilizando sentencias LPRINT y la función CHR$ (*argumento*) de la forma siguiente:

LPRINT CHR$(14)	Activación caracteres ampliados.
LPRINT CHR$(20)	Cancela caracteres ampliados.
LPRINT CHR$(15)	Selecciona la impresión condensada. (132 caracteres por línea en FX-85, 233 caracteres por línea en FX-105).
LPRINT CHR$(18)	Cancela la impresión condensada.
LPRINT CHR$(27); CHR$(69)	Activa impresión en negrita.
LPRINT CHR$(27); CHR$(70)	Cancela impresión en negrita.
LPRINT CHR$(27); CHR$(71)	Activa impresión en doble pasada.
LPRINT CHR$(27); CHR$(72)	Anula impresión en doble pasada.
LPRINT CHR$(27); CHR$(83); CHR$(0)	Activa el modo escritura en superíndice.
LPRINT CHR$(27); CHR$(83); CHR$(1)	Activa el modo escritura en subíndice.
LPRINT CHR$(27); CHR$(84)	Cancela escritura superíndice/subíndice.
LPRINT CHR$(27); CHR$(45); CHR$(1)	Selecciona el modo de escritura subrayado.
LPRINT CHR$(27); CHR$(45); CHR$(0)	Cancela el modo de escritura subrayado.
LPRINT CHR$(27); CHR$(52)	Selecciona el modo de escritura cursiva.
LPRINT CHR$(27); CHR$(53)	Cancela el modo de escritura cursiva.

Ejemplos

```
10 REM pruebas de tipos de letras
20 LPRINT "impresión normal"
30 LPRINT CHR$(14) "impresión ampliada "
40 ' se cancela la impresión ampliada  sin necesidad
50 ' de codigo de control
60 LPRINT CHR$(15); " impresión comprimida "
70 LPRINT CHR$(18) ; " desactivación impresión comprimida"
80 LPRINT CHR$(27)"G" ; "impresión doble pasada "
90 LPRINT CHR$(27) "H" ; " cancelación impresión doble pasada "
100 LPRINT CHR$(27); CHR$(83) ; CHR$(0) ;"superindice 234 "
110 LPRINT CHR$(27);CHR$(84) ;" desactivacoón superindice"
120 LPRINT "escritura normal"
130 LPRINT CHR$(27);CHR$(83); CHR$(1) ; " subindice  234"
140 LPRINT CHR$(27);CHR$(84)
150 LPRINT CHR$(27);CHR$(69); "escritura en negrita"
160 LPRINT CHR$(27); CHR$(70); " desactivación de la escritura en negrita"
170 LPRINT CHR$(27); CHR$(71) ;" otro método de escritura en doble pasada"
180 LPRINT CHR$(27); CHR$(72) ;" otro método de desactivación en dble.pa."
190 LPRINT "fin de la prueba "

run
```

```
impresión normal
impresión ampliada
impresión comprimida
 desactivación impresión comprimida
impresión doble pasada
 cancelación impresión doble pasada
superindice 234
 desactivacoón superindice
escritura normal
 subindice  234

escritura en negrita
 desactivación de la escritura en negrita
 otro método de escritura en doble pasada
 otro método de desactivación en dble.pa.
fin de la prueba
```

```
10 REM prueba de espaciado de líneas
20 '
30 '
40 LPRINT " espaciado entre líneas normal, 1/6 " "
50 LPRINT " prueba de demostración "
60 LPRINT CHR$(27) "0"      ' activación espaciado a 1/8 "
70  LPRINT " espaciado entre líneas  1/8"
80 LPRINT " prueba de demostración "
90 LPRINT CHR$(27) "1"      ' activación espaciado a 7/72"
100 LPRINT "prueba de espacio entre líneas más comprimido, 7/72"
110 LPRINT "sigue la prueba"
120 LPRINT CHR$(27) "2" ; " espaciado normal "
130 LPRINT "fin de la prueba"

run
```

```
espaciado entre líneas normal, 1/6
prueba de demostración

espaciado entre líneas  1/8
prueba de demostración

prueba de espacio entre líneas más comprimido, 7/72
sigue la prueba
espaciado normal
fin de la prueba
```

```
10 REM nueva prueba
20 '
30 LPRINT CHR$(14); "comienzo de la prueba"
40 LPRINT "escritura normal"
50 LPRINT CHR$(27);CHR$(87);CHR$(1); " la activación de la escritura ampliada se
 conservará durante mas de una línea , debido a que se ha activado este tipo de
letra con los codigos de control 27, 87 y 1"
60 LPRINT CHR$(27) ; CHR$(87); CHR$(0)
70 '
80 ' mezcla de caracteres comprimidos y espaciado de líneas 7/72"
90 LPRINT CHR$(27) "1" CHR$(15)
100 LPRINT "escritura comprimida y con espaciado de líneas 7/72 , la prueba dura
rá hasta que se cancelen la escritura comprimida y el espaciado de línea se pong
a de nuevo a 1/6 o bien a 1/8 "
110 LPRINT CHR$(27) "2"          ' desactivación espaciado 7/72
120 LPRINT "todavía debe desactivar la escritura comprimida"
130 LPRINT CHR$(18)
140 LPRINT "ahora si que ha terminado la prueba"
RUN
```

```
comienzo de la prueba
escritura normal

 la activación de la escritura ampliada
se conservará durante mas de una línea ,
 debido a que se ha activado este tipo d
e letra con los codigos de control 27, 8
7 y 1

escritura comprimida y con espaciado de líneas 7/72 , la prueba durará hasta que
 se cancelen la escritura comprimida y el espaciado de línea se ponga de nuevo a
 1/6 o bien a 1/8

todavía debe desactivar la escritura comprimida

ahora si que ha terminado la prueba
comienzo de la prueba

 adelante
sigue la prueba
```

```
10 REM prueba de tabulaciones
20 '
30 '
40 LPRINT "comienzo de la prueba"
50 LPRINT CHR$(11) ; " adelante"
60 LPRINT "sigue la prueba"
70 LPRINT "Instituto";TAB(25);"Cervantes";TAB(40) "Alcalá"
RUN
```

```
comienzo de la prueba

 adelante
sigue la prueba
Instituto                    Cervantes        Alcalá
```

```
100 LPRINT CHR$(27);"N"
110 B$="Instituto Miguel de Cervantes"
120 B1$="INSTITUTO MIGUEL DE CERVANTES"
130 LPRINT 1; B$
140 LPRINT
150 LPRINT 2; B1$
160 LPRINT CHR$(27); "E"
170 LPRINT 3; B1$
180 LPRINT CHR$(27); "P"
190 LPRINT 4; B1$
200 LPRINT CHR$(27); "Q"
210 LPRINT 5; B1$
220 LPRINT CHR$(27); "N"; CHR$(14)
230 LPRINT 6; B1$

RUN
```

```
1 Instituto Miguel de Cervantes

2 INSTITUTO MIGUEL DE CERVANTES

3 INSTITUTO MIGUEL DE CERVANTES

4 INSTITUTO MIGUEL DE CERVANTES

5 INSTITUTO MIGUEL DE CERVANTES

6 INSTITUTO MIGUEL DE CERVANTES
```

Impresoras conectadas a computadoras con versiones de Microsoft BASIC

Las órdenes de impresión son las ya mencionadas sentencias LPRINT dentro de programas y LLIST para obtener listados totales o parciales.

```
10 PRINT "Demo1"        ' visualización en pantalla
20 LPRINT "Demox"       ' impresión en impresora
```

Impresión en computadoras Commodore/Atari

Commodore BASIC y Atari BASIC requieren que la salida a la impresora sea dirigida a través de un canal o fichero. Estos dialectos requieren una sentencia OPEN para crear un fichero, de modo que toda la información impresa en el fichero (o canal) sale hacia la impresora. En este caso la sentencia PRINT se modifica con el formato PRINT#, en lugar de LPRINT como Microsoft BASIC.

```
10 OPEN 1,4                       apertura del fichero de la impresora
20 PRINT#1, "Demostración "
30 PRINT#1, CHR$(14); "doble ancho"
40 PRINT#1,CHR$(14);"ancho";CHR$(15);"normal"
```

Impresoras Apple II con Applesoft BASIC

Las sentencias utilizadas en este caso son PR#1 para conmutar la salida a impresora y PR#0 devuelve la salida a la pantalla.

```
10 PRINT CHR$(4); PR#1 : REM activar impresora
20 C$=ESC$ + "q" : REM escritura comprimida
30 PRINT C$ : "comprimida, 120 caracteres por línea"
```

Activación de impresoras bajo control de los sistemas operativos MS-DOS y CP/M

La orden CTRL-P activa la impresora desde el sistema operativo, haciendo que se produzca un eco en la misma o, lo que es igual, todo lo que se escriba en pantalla se imprimirá en la impresora, hasta que se desactive de nuevo con la orden CTRL-P.

En computadoras BBC, desde su BASIC se puede seleccionar la impresora de igual modo; es decir, con las órdenes VDU o CTRL-B.

Escritura en impresoras conectadas a computadoras Amstrad

Las computadoras Amstrad consideran a la impresora como el canal 8 de salida. Por consiguiente las sentencias de impresión son:

`PRINT #8`	*imprime en impresora*
`LIST #8`	*orden de listado en impresora*

```
10 PRINT "Demo1"
20 PRINT #8,"Escritura en impresora"
30 PRINT "Demo3"
```

`RUN`	
`Demo1`	*visualización en pantalla*
`Escritura en impresora`	*se imprime en impresora*
`Demo3`	*visualización en pantalla*

III

Tablas de conversión

Este apéndice contiene las siguientes tablas de referencia:

- Conversión de hexadecimal-binario.
- Conversión de enteros hexadecimal-decimal.
- Potencias de dos.
- Constantes matemáticas.
- Potencias de dieciséis.
- Potencias de diez.

TABLA DE CONVERSION HEXADECIMAL A BINARIO

Utilice la tabla siguiente para conversión de números hexadecimales en el intervalo 0-0F y los números binarios en el intervalo 0000-1111.

Se convierten números binarios más grandes a números hexadecimales por conversión de cuatro dígitos a la vez, operando de derecha a izquierda. Si hay menos de cuatro dígitos en el grupo más a la izquierda, añada ceros a la izquierda. He aquí un ejemplo:

$$100101_2 = \underbrace{\underbrace{0010}_{2}\,\underbrace{0101}_{5}}_{25_{16}}{}_2$$

Se convierten números hexadecimales más grandes de 0F a binario, un dígito cada vez. He aquí un ejemplo:

$$37_{16} \to 3 \quad 7$$

$$\underbrace{\overbrace{0011}^{3}\ \overbrace{0111}^{7}}_{00110111_2}$$

Hexadecimal	*Binario*	*Decimal*
00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
0A	1010	10
0B	1011	11
0C	1100	12
0D	1101	13
0E	1110	14
0F	1111	15

CONVERSION DE ENTEROS HEXADECIMAL-DECIMAL

La tabla siguiente proporciona conversiones directas entre enteros hexadecimales en el margen de 0-FFF y enteros decimales en el margen de 0-4095. Para conversión de enteros más grandes, los valores de la tabla se pueden añadir a las siguientes cifras:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Las partes decimales se pueden convertir en fracciones decimales de la siguiente forma:

1. Exprese la parte decimal como el producto de un entero por 16^{-n}, en donde n es el número de lugares significativos hexadecimales a la derecha del punto hexadecimal.

$$0.\ CA9BF3_{16} = CA9\ BF3_{16} \times 16^{-6}$$

2. Calcule el equivalente decimal del entero hexadecimal.

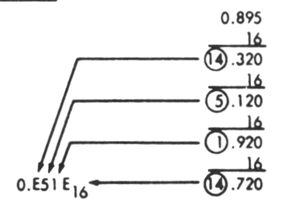
$$CA9\ BF3_{16} = 13\ 278\ 195_{10}$$

3. Multiplique el equivalente decimal por 16^{-n}

$$\begin{array}{r} 13\ 278\ 195 \\ \times\ 596\ 046\ 448 \times 10^{-16} \\ \hline 0.791\ 442\ 096_{10} \end{array}$$

Las partes decimales se pueden convertir en partes hexadecimales por multiplicaciones sucesivas de la parte decimal por 16_{10}. Después de cada multiplicación, la parte entera se extrae para formar una parte hexadecimal construyéndola a la derecha del punto hexadecimal. Sin embargo, ya que se usa aritmética decimal en esta conversión, la parte de cada producto se debe convertir en números hexadecimales.

Ejemplo: Convertir 0.895_{10} en su equivalente hexadecimal



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

CONVERSION DE ENTEROS HEXADECIMAL-DECIMAL. *(Continuación)*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

CONVERSION DE ENTEROS HEXADECIMAL-DECIMAL. *(Continuación)*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1562	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

CONVERSION DE ENTEROS HEXADECIMAL-DECIMAL. *(Continuación)*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

CONVERSION DE ENTEROS HEXADECIMAL-DECIMAL. ***(Continuación)***

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

CONVERSION DE ENTEROS HEXADECIMAL-DECIMAL. *(Continuación)*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

POTENCIAS DE DIECISEIS

16^n	n	16^{-n}	
1	0	0.10000 00000 00000 00000	× 10
16	1	0.62500 00000 00000 00000	× 10^{-1}
256	2	0.39062 50000 00000 00000	× 10^{-2}
4 096	3	0.24414 06250 00000 00000	× 10^{-3}
65 536	4	0.15258 78906 25000 00000	× 10^{-4}
1 048 576	5	0.95367 43164 06250 00000	× 10^{-6}
16 777 216	6	0.59604 64477 53906 25000	× 10^{-7}
268 435 456	7	0.37252 90298 46191 40625	× 10^{-8}
4 294 967 296	8	0.23283 06436 53869 62891	× 10^{-9}
68 719 476 736	9	0.14551 91522 83668 51807	× 10^{-10}
1 099 511 627 776	10	0.90949 47017 72928 23792	× 10^{-12}
17 592 186 044 416	11	0.56843 41886 08080 14870	× 10^{-13}
281 474 976 710 656	12	0.35527 13678 80050 09294	× 10^{-14}
4 503 599 627 370 496	13	0.22204 46049 25031 30808	× 10^{-15}
72 057 594 037 927 936	14	0.13877 78780 78144 56755	× 10^{-16}
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721	× 10^{-18}

POTENCIAS DE DIEZ
(Convertidas en valores hexadecimales)

10^n	n	10^{-n}	
1	0	1.0000 0000 0000 0000	
A	1	0.1999 9999 9999 999A	
64	2	0.28F5 C28F 5C28 F5C3	× 16^{-1}
3E8	3	0.4189 374B C6A7 EF9E	× 16^{-2}
2710	4	0.68DB 8BAC 710C B296	× 16^{-3}
1 86A0	5	0.A7C5 AC47 1B47 8423	× 16^{-4}
F 4240	6	0.10C6 F7A0 B5ED 8D37	× 16^{-4}
98 9680	7	0.1AD7 F29A BCAF 4858	× 16^{-5}
5F5 E100	8	0.2AF3 1DC4 6118 73BF	× 16^{-6}
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC	× 16^{-7}
2 540B E400	10	0.6DF3 7F67 5EF6 EADF	× 16^{-8}
17 4876 E800	11	0.AFEB FF0B CB24 AAFF	× 16^{-9}
E8 D4A5 1000	12	0.1197 9981 2DEA 1119	× 16^{-9}
918 4E72 A000	13	0.1C25 C268 4976 81C2	× 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604	× 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D	× 16^{-12}
23 8652 6FC1 0000	16	0.734A CA5F 6226 F0AE	× 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449	× 16^{-14}
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1	× 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35	× 16^{-15}

POTENCIAS DE DOS

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 661 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 235 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

CONSTANTES MATEMATICAS

Constante	Valor decimal	Valor hexadecimal
n	3.14159 26535 89793	3.243F 6A89
n^{-1}	0.31830 98861 83790	0.517C C187
$\sqrt{n}$	1.77245 38509 05516	1.C5BF 891C
$\ln\pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.87E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10}e$	0.43429 44819 03252	0.6F2D EC55
$\log_2 e$	1.44269 50408 88963	1.7154 7653
Y	0.57721 56649 01533	0.93C4 67E4
lnY	−0.54953 93129 81645	−0.8CAE 98C1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
ln2	0.69314 71805 59945	0.8172 17F8
$\log_{10}2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.298B 075C
ln10	2.30258 40929 94046	2.4D75 3777

IV

Subrutinas útiles

BUSQUEDA SECUENCIAL

El objetivo de esta subrutina es realizar la búsqueda de una palabra (elemento) en una lista almacenada en un fichero de sentencias DATA; aunque dicho fichero podría considerarse almacenado en un disco.

```
10 REM busqueda secuencial
20 INPUT "palabra a buscar :"; P$
30 '
40 '
50 READ N
60 DIM A$(N)
70 FOR I=1 TO N
80     READ A$(I)
90     IF P$ = A$(I) THEN 150
100 NEXT I
110 '
120 '
130 J=0 : PRINT A$ ; "no se encuentra en la lista "
140 GOTO 200
150 PRINT A$; "se encuentra situada en el puesto "; I
160 '
170 '
180 DATA 12
190 DATA manzana, pera, melón, uva, melocotón, naranja, fresa, paraguaya,
    chirimolla, platano, piña, sandia
200 END
```

```
Ok
run
palabra a buscar :? uva
se encuentra situada en el puesto  4
Ok
run
palabra a buscar :? pera
se encuentra situada en el puesto  2
Ok
run
palabra a buscar :? cereza
no se encuentra en la lista
Ok
run
palabra a buscar :? chirimolla
se encuentra situada en el puesto  9
Ok
```

CLASIFICACION POR ORDEN ALFABETICO/DECRECIENTE

```
 10 REM ** Clasificacion alfabetica **
 20 REM
 30 REM
 40 REM
 50 PRINT "****************************
 60 PRINT "* Clasificacion alfabetica *
 70 PRINT "*    en orden creciente    *
 80 PRINT "*       de palabras        *
 90 PRINT "****************************
 100 PRINT
 110 REM * Entrada de datos *
 120 PRINT "Numero de palabras a clasificar";
 130 INPUT N
 150 PRINT
 160 PRINT "Introduzca los datos"
 170 FOR I=1 TO N
 180 INPUT A$(I)
 200 NEXT I
 210 REM * Comienza la clasificacion *
 220 C=0
 230 FOR I=1 TO N-1
 240 IF A$(I)<=A$(I+1) THEN 290
 250 B$=A$(I)
 260 A$(I)=A$(I+1)
 270 A$(I+1)=B$
 280 C=1
 290 NEXT I
 300 IF C=1 THEN 220
 310 PRINT
 320 PRINT
 330 REM * Salida resultados *
 340 FOR I=1 TO N
 350 PRINT A$(I)
 360 NEXT I
 370 END
```

```
RUN
****************************
* Clasificacion alfabetica *
*    en orden creciente    *
*       de palabras        *
****************************

Numero de palabras a clasificar? 6

Introduzca los datos
?Juani
?Luis
?Alfonso
?Angel
?Rosa
?Javier

Alfonso
Angel
Javier
Juani
Luis
Rosa
```

CLASIFICACION POR ORDEN ALFABETICO/CRECIENTE

```
10 REM ** Clasificacion alfabetica **
20 REM
30 REM
40 REM
50 PRINT "****************************
60 PRINT "* Clasificacion alfabetica *
70 PRINT "*   en orden decreciente   *
80 PRINT "*        de palabras       *
90 PRINT "****************************
100 PRINT
110 REM * Entrada de datos *
120 PRINT "Numero de palabras a clasificar";
130 INPUT N
150 PRINT
160 PRINT "Introduzca los datos"
170 FOR I=1 TO N
180 INPUT A$(I)
200 NEXT I
210 REM * Comienza la clasificacion *
220 C=0
230 FOR I=1 TO N-1
240 IF A$(I)>=A$(I+1) THEN 290
250 B$=A$(I)
260 A$(I)=A$(I+1)
270 A$(I+1)=B$
280 C=1
290 NEXT I
300 IF C=1 THEN 220
310 PRINT
320 PRINT
330 REM * Salida resultados *
340 FOR I=1 TO N
350 PRINT A$(I)
360 NEXT I
370 END
```

```
****************************
* Clasificacion alfabetica *
*   en orden decreciente   *
*        de palabras       *
****************************

Numero de palabras a clasificar? 5

Introduzca los datos
?Juani
?Luis
?Alfonso
?Angel
?Rosa


Rosa
Luis
Juani
Angel
Alfonso
```

CLASIFICACION NUMERICA/DECRECIENTE

```
10 REM ** Clasificacion numerica **
20 REM
30 REM
40 REM
50 PRINT "*************************"
60 PRINT "* Clasificacion numerica *"
70 PRINT "*  en orden decreciente  *"
80 PRINT "*      de numeros        *"
90 PRINT "*************************"
100 PRINT
110 REM * Entrada de datos *
120 PRINT "Cuantos numeros";
130 INPUT N
140 PRINT
150 PRINT "Introduzca los datos"
160 FOR I=1 TO N
170 INPUT A(I)
180 NEXT I
190 REM * Comienza la clasificacion *
200 C=0
210 FOR I=1 TO N-1
220 IF A(I)>= A(I+1) THEN 270
230 B=A(I)
240 A(I)=A(I+1)
250 A(I+1)=B
260 C=1
270 NEXT I
280 IF C=1 THEN 200
290 PRINT
300 PRINT
310 REM * Salida resultados *
320 FOR I=1 TO N
330 PRINT A(I)
340 NEXT I
350 END

RUN
*************************
* Clasificacion numerica *
*  en orden decreciente  *
*      de numeros        *
*************************

Cuantos numeros
Introduzca los datos
? 859
? 456
? 321
? 956
? 357


 956
 859
 456
 357
 321
```

CLASIFICACION NUMERICA/DECRECIENTE

```
10 REM ** Clasificacion numerica **
20 REM
30 REM
40 REM
50 PRINT "*************************"
60 PRINT "* Clasificacion numerica *"
70 PRINT "*   en orden creciente   *"
80 PRINT "*      de numeros        *"
90 PRINT "*************************"
100 PRINT
110 REM * Entrada de datos *
120 PRINT "Cuantos numeros";
130 INPUT N
140 PRINT
150 PRINT "Introduzca los datos"
160 FOR I=1 TO N
170 INPUT A(I)
180 NEXT I
190 REM * Comienza la clasificacion *
200 C=0
210 FOR I=1 TO N-1
220 IF A(I)<= A(I+1) THEN 270
230 B=A(I)
240 A(I)=A(I+1)
250 A(I+1)=B
260 C=1
270 NEXT I
280 IF C=1 THEN 200
290 PRINT
300 PRINT
310 REM * Salida resultados *
320 FOR I=1 TO N
330 PRINT A(I)
340 NEXT I
350 END

RUN
*************************
* Clasificacion numerica *
*   en orden creciente   *
*      de numeros        *
*************************

Cuantos numeros
Introduzca los datos
? 46
? 789
? 951
? 258
? 650


 46
 258
 650
 789
 951
```

SUBRUTINA AGENDA

La siguiente subrutina permite disponer de una AGENDA de direcciones con indicación de nombres, domicilios y profesiones. Permite la localización de una persona incluida en la misma mediante cualquier dato de la ficha correspondiente.

Ha sido desarrollado en una impresora de 80 columnas, aunque las personas que sólo dispongan de pantalla bastará con que modifiquen adecuadamente las líneas 360, 350, 440, 510, 580 y 590.

La longitud de los campos ha sido también fijada, pero no presenta problema su variación. En nuestro caso la selección ha sido la siguiente:

Apellidos:	20 caracteres
Nombre:	10 caracteres
Profesión:	12 caracteres
Teléfono:	12 caracteres
Ciudad:	15 caracteres
D Postal:	2 caracteres

Asimismo hemos de destacar que los datos del fichero los hemos recogido en sentencias READ/DATA (la palabra fin de fichero en la sentencia DATA es precisamente «fin»). Por economía de tiempo y espacio hemos realizado las pruebas sólo con cuatro personas, pero evidentemente los lectores pueden ampliar el fichero tanto como se lo permita la memoria de su computadora. La potencia del programa aumentaría si se puede utilizar ficheros de casete o disquete con las adecuadas modificaciones.

El programa, como se comprueba en las diferentes ejecuciones, permite la visualización completa de la agenda o la búsqueda de cualquier persona contenida en ella mediante un campo concreto: nombre, apellidos, teléfono, profesión, ciudad o distrito postal. Bastará para ello solicitarlo así, es decir, introducir el nombre, apellido, teléfono, etc., tras lo cual la computadora responderá con la lista correspondiente al campo solicitado o bien con una línea en blanco en el caso de que el dato solicitado no estuviese archivado en la agenda.

El orden de ejecuciones ha sido el siguiente:

1. Se ha introducido el apellido MARTINEZ y como éste no existe nos ha entregado una línea en blanco.
2. Se ha tecleado A y se ha listado la agenda completa.
3. Se ha introducido MADRID y nos ha entregado la lista de todas las personas con domicilio en MADRID.
4. Al pulsar la palabra médico se listan las personas con esa profesión.
5. Lista de apellidos de personas de la agenda apellidadas GARCIA.
6. Lista de personas cuyo número de teléfono es 91/6285215.

Resumiendo, las líneas 140 a 190 listan e imprimen el menú de selección. De modo que si deseamos listar la agenda completa pulsaremos A, para terminar el programa la letra F, y si se desea buscar las personas por alguno de los códigos bastará con introducir el mismo. Por ejemplo, si se desea ver cuántas personas hay de profesión ABOGADO será preciso escribir esta palabra en el menú de selección.

Evidentemente este programa puede ser mejorado y ampliado para adaptarlo a las necesidades específicas del lector. Nuestra intención ha sido proporcionar una herramienta básica de trabajo que permita el desarrollo específico de aplicaciones concretas.

```
10 REM ** Agenda **
20 REM **        **
30 REM **        **
40 PRINT "**************"
50 PRINT "*   Agenda   *"
60 PRINT "**************"
70 PRINT
80 PRINT "Codigo de seleccion"
90 PRINT "Se puede buscar"
100 PRINT "de las siguientes"
110 PRINT "formas"
120 PRINT
130 PRINT "Nombre"
140 PRINT "Apellidos"
150 PRINT "Telefono"
160 PRINT "Ciudad"
170 PRINT "Profesion"
180 PRINT "Distrito postal"
190 PRINT
200 PRINT
210 PRINT "A -- Agenda"
220 PRINT "F -- Fin"
230 PRINT
240 F$="FIN"
250 INPUT S$
260 IF S$="F" THEN 650
270 RESTORE
280 A$="A"
290 IF S$="A" THEN 450
300 PRINT "Apellidos            Nombre    Profesion   Telefono  Ciudad  D.P."
310 PRINT "------------------------------------------------------------------"
320 PRINT
330 READ N$, A$, P$, T$, C$, D$
340 IF (N$=S$) OR (A$=S$) OR (T$=S$) THEN 390
350 IF (P$=S$) OR (C$=S$) OR (D$=S$) THEN 390
360 IF (N$=F$) OR (A$=F$) OR (T$=F$) THEN 420
370 IF (P$=F$) OR (C$=F$) OR (D$=F$) THEN 640
380 GOTO 330
390 PRINT N$; TAB(21); A$; TAB(31); P$; TAB(43); T$; TAB(58); C$; TAB(72); D$
400 PRINT
410 GOTO 330
420 PRINT
430 PRINT
440 GOTO 650
450 RESTORE
460 PRINT "Apellidos            Nombre    Profesion   Telefono  Ciudad  D.P."
470 PRINT
480 PRINT "------------------------------------------------------------------"
490 PRINT
500 FOR I=1 TO 5
510 READ N$, A$, P$, T$, C$, D$
520 IF N$="FIN" THEN GOTO 80
530 PRINT N$; TAB(21); A$; TAB(31); P$;
540 PRINT TAB(43); T$; TAB(58); C$; TAB(72); D$
550 NEXT I
560 PRINT
570 PRINT "----------------------------------------------------------------"
580 FOR W=1 TO 2000 : NEXT W
590 GOTO 500
600 DATA Garcia, Luis, Abogado, 91/6585215, Madrid, 28033
610 DATA Jimenez, Manuel, Medico, 91/4331258, Madrid, 28014
620 DATA Lopez, Jose, Fontanero, 926/323254, Soria, 0
```

```
630 DATA Garcia, Jose, Medico, 053/234567, Avila, 0
640 DATA FIN, FIN, FIN, FIN, FIN, FIN
650 PRINT F$
660 END


RUN

***************
*    AGENDA   *
***************

CODIGO DE SELECCION
SE PUEDE BUSCAR
DE LAS SIGUIENTES
FORMAS

NOMBRE
APELLIDOS
TELEFONO
CIUDAD
PROFESION
DISTRITO POSTAL

A - AGENDA
F - FIN




APELLIDOS     NOMBRE     PROFESION     TELEFONO     CIUDAD     D.P
------------------------------------------------------------------




APELLIDOS     NOMBRE     PROFESION     TELEFONO     CIUDAD     D.P.
-------------------------------------------------------------------
GARCIA        LUIS       ABOGADO       91/6585215   MADRID      23
JIMENEZ       MANUEL     MEDICO        91/4331258   MADRID      14
LOPEZ         JOSE       FONTANERO     926/323254   SORIA        0
GARCIA        JOSE       MEDICO        053/234567   AVILA        0




APELLIDOS     NOMBRE     PROFESION     TELEFONO     CIUDAD     D.P.
-------------------------------------------------------------------
GARCIA        LUIS       ABOGADO       91/6585215   MADRID      23
JIMENEZ       MANUEL     MEDICO        91/4331258   MADRID      14
```

```
APELLIDOS      NOMBRE      PROFESION      TELEFONO       CIUDAD      D.P.
-------------------------------------------------------------------------
JIMENEZ        MANUEL      MEDICO         91/4331258     MADRID        14
GARCIA         JOSE        MEDICO         053/234567     AVILA          0
```

```
APELLIDOS      NOMBRE      PROFESION      TELEFONO       CIUDAD      D.P.
-------------------------------------------------------------------------
GARCIA         LUIS        ABOGADO        91/6585215     MADRID        23
GARCIA         JOSE        MEDICO         053/234567     AVILA          0
```

```
APELLIDOS      NOMBRE      PROFESION      TELEFONO       CIUDAD      D.P.
-------------------------------------------------------------------------
GARCIA         LUIS        ABOGADO        91/6585215     MADRID        23
```

SUBRUTINA DE OBTENCION DE NOTAS MEDIAS POR ALUMNO Y ASIGNATURA DE UN DETERMINADO CURSO DE ENSEÑANZA

El siguiente programa nos permite la introducción de las notas de un determinado curso, pudiendo elegir el número de alumnos así como de asignaturas, ya que las variables que lo representan se dimensionan automáticamente tras la introducción de sus correspondientes valores. El nombre de las asignaturas se introduce con abreviaturas de tres letras.

La introducción de las notas se hace alumno por alumno escribiendo todas sus notas del mes, trimestre o curso.

El programa ha sido desarrollado para la microcomputadora IBM PC y permite su presentación a voluntad en pantalla o en impresora IBM o Epsón.

```
100 CLS
110 REM Entrada de datos
130 PRINT : INPUT " (Quiere datos por impresora)"; A$
140 PRINT:PRINT
150 INPUT"Curso/Seccion"; C$
160 INPUT "Numero de alumnos"; N
170 INPUT "Numero de asignaturas"; A
180 DIM NC$(N) : DIM M(N) : DIM C(N,A)
190 CLS:PRINT "Asignaturas (Max 3 letras):"
200 FOR J=1 TO A
210 PRINT J; ".-";
220 INPUT A$(J)
230 NEXT J
240 REM Entrada de datos
260 FOR I=1 TO N
270 CLS:PRINT:INPUT "Nombre completo (24 letras max)"; NC$(I):PRINT
280 PRINT "Introduzca todas las calificaciones del alumno":PRINT
290 FOR J=1 TO A
300 PRINT A$(J); ".-";
310 INPUT C(I,J)
320 M(I)=M(I)+C(I,J)
330 NEXT J
340 M(I)=M(I)/A
350 M(I)=INT (100*M(I)+.5)/100
360 NEXT I
370 FOR J=1 TO A
380 FOR I=1 TO N
390 X(J)=X(J)+C(I,J)
400 NEXT I
410 MC=MC+X(J)
420 X(J)=INT(.5+(10*(X(J)/N)))/10
430 NEXT J
440 MC=MC/(N*A)
450 MC=FIX(.5+(100*MC))/100
460 CLS:PRINT "Instituto Miguel de Cervantes"
470 IF A$="S" THEN LPRINT CHR$(14);"Instituto Miguel de Cervantes"
480 IF A$="S" THEN LPRINT : LPRINT "------------------------------":LPRINT:LPRINT
 CHR$(15)
500 PRINT "-----------------------------"
510 PRINT "(Curso/Seccion) "; C$
520 IF A$="S" THEN LPRINT CHR$(14);"(Curso/Seccion)"; C$; CHR$(15):LPRINT:LPRINT
530 PRINT "Nombre "TAB(11)A$(1); TAB(16)A$(2); TAB(21)A$(3); TAB(26)"Media"
540 IF A$="S" THEN LPRINT "Nombre y apellidos ";
550 FOR J=1 TO A
560 IF A$="S" THEN LPRINT TAB(26+(J-1)*5) A$(J);
570 NEXT J
580 IF A$="S" THEN LPRINT TAB(73)"Media"
```

```
590 IF A$="S" THEN LPRINT " - - - - - - - - - - - - ";
600 FOR J=1 TO A
610 IF A$="S" THEN LPRINT " - - - - - - - ";
615 NEXT J
620 IF A$="S" THEN LPRINT TAB(73) " - - - - - - -":LPRINT
630 PRINT
640 FOR I=1 TO N
650 PRINT NC$(I);
660 IF A$="S" THEN LPRINT NC$(I);
670 FOR J=1 TO A
680 PRINT TAB(10+(J-1)*5)C(I,J);
690 IF A$="S" THEN LPRINT TAB(25+(J-1)*5) C(I,J);
700 NEXT J
710 PRINT TAB(25)M(I)
720 IF A$="S" THEN LPRINT TAB(73)M(I)
730 NEXT I
740 PRINT " - - - - - - - - - - - - - - - - -"
750 IF A$="S" THEN LPRINT " - - - - - - - - - - - - -"
760 FOR J=1 TO A
770 IF A$="S" THEN LPRINT " - - - - - - -"; : NEXT J
780 IF A$="S" THEN LPRINT TAB(73) "- - - - - - -"
790 PRINT "Medias";
800 IF A$="S" THEN LPRINT "Medias";
810 FOR J=1 TO A
820 PRINT TAB(10+(J-1)*5)X(J);
830 IF A$="S" THEN LPRINT TAB(25+(J-1)*5)X(J);
840 NEXT J
850 PRINT TAB(25) MC
860 IF A$="S" THEN LPRINT TAB(73) MC
870 END


RUN
(Quiere datos por impresora S/N)? S
Curso / Seccion? SDT Y DLO

Numero de alumnos? 4
Numero de asignaturas? 4

Asignaturas (Max 3 letras)
 1 .-DIB
 2 .-CAL
 3 .-ALG
 4 .-ELE
Nombre Completo (24 letras max)?Jose Garcia Perez

Introduzca todas las calificaciones del alumno

DIB.-? 6.7
CAL.-? 8.5
ALG.-? 4.5
ELE.-? 5.5
Nombre Completo (24 letras max)?Antonio Diaz Merchan


Introduzca todas las calificaciones del alumno

DIB.-? 5.6
CAL.-? 6.7
ALG.-? 5.5
ELE.-? 8.5
Nombre Completo (24 letras max)?Javier Sanz Gomez
```

```
Introduzca todas las calificaciones del alumno

DIB.-? 5.1
CAL.-? 5.4
ALG.-? 7.6
ELE.-? 6.9
Nombre Completo (24 letras max)?Tomas Rollo Pita

Introduzca todas las calificaciones del alumno

DIB.-? 6.5
CAL.-? 7.2
ALG.-? 5.6
ELE.-? 6.9


Instituto Miguel de Cervantes
-----------------------------

(Curso/Seccion)COU / A

Nombre y apellidos       DIB  CAL  ALG  ELE            Media
- - - - - - - - - - - -  - - - - - -  - - - - - - - - - - - - - - - - -
Jose Garcia Perez        6.7  8.5  4.5  5.5              6.3
Antonio Diaz Merchan     5.6  6.7  5.5  8.5              6.58
Javier Sanz Gomez        5.1  5.4  7.6  6.9              6.25
Tomas Rollo Pita         6.5  7.2  5.6  6.9              6.55
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Medias                   6    7    5.8  7               6.42
```

METODO DE INTERPOLACION DE NEWTON

En numerosas ocasiones, sobre todo en cálculos técnicos y científicos, se necesitan interpolaciones para obtener valores de una función de acuerdo con los correspondientes valores. Aunque el método exige un estudio matemático riguroso, se puede aplicar en la mayoría de los casos en que se trabaja con magnitudes físicas medibles como tablas de presiones, temperaturas, trigonométricas, de tiro de cañones, etc.

Con el programa de interpolación se consigue alojar las tablas mencionadas en muy poco espacio de memoria recurriendo cada vez que se necesite un valor determinado a realizar la correspondiente interpolación.

El método se basa en tomar *n* valores conocidos de la variable x, x_0, x_1, x_2,..., x_n de tal forma que $x_1-x_0=x_2-x_1=\ldots=x_n-x_{n-1}=K$. Sean $y_0, y_1, y_2, \ldots, y_n$ los correspondientes valores de y. Se obtienen coeficientes $C_1, C_2, C_3, \ldots, C_n$; $K_1, K_2, K_3, \ldots, K_n$ que responden a una fórmula concreta, y, una vez obtenidos, el valor y que corresponde a cualquier valor x que esté comprendido en el intervalo x_0-x_n, responde a la ecuación:

$$y=K_n \cdot x^{n-1} \cdots +K_4 \cdot x^3+K_3 \cdot x^2+K_2 \cdot x+K_1$$

El ejemplo práctico realizado en una computadora IBM PC obtiene la tabla de funciones trigonométricas seno, coseno y tangente de grado en grado, partiendo de los valores conocidos de dichas funciones para 0º, 22,5º, 45º, 67,5º, 90º, aunque en la tabla de salida sólo se obtiene hasta 45º debido a que a partir de este punto como la función tangente tiende asintóticamente al infinito deja de cumplirse la Ley de Newton y la interpolación no es válida.

Datos del programa:

$$x_0, x_1, x_2, x_3, x_4, \ldots$$
$$y_0, y_1, y_2, y_3, y_4, \ldots$$

VALORES DE LOS COEFICIENTES EN FUNCION DE $H=x_1-x_0=x_2-x_1=\ldots$

$$C_1=y_0$$
$$C_2=(y_1-y_0)/H$$
$$C_3=(y_2-2 \cdot y_1+y_0)/(2 \cdot H^2)$$
$$C_4=(y_3-3 \cdot y_2+3 \cdot y_1-y_0)/(6 \cdot H^3)$$
$$C_5=(y_4-4 \cdot y_3+6 \cdot y_2-4 \cdot y_1+y_0)/(24 \cdot H^4)$$
$$K_1=C_1-x_0 \cdot (C_2-x_1 \cdot (C_3-x_2 \cdot (C_4-C_5 \cdot x_3)))$$
$$K_2=C_2-C_3 \cdot (x_0+x_1)+(C_4-C_5 \cdot x_3) \cdot (x_0 \cdot x_1+x_0 \cdot x_2+x_1 \cdot x_2)-C_5 \cdot x_0 \cdot x_1 \cdot x_2$$
$$K_3=C_3-C_4 \cdot (x_0+x_1+x_2)+C_5 \cdot (x_0 \cdot (x_1+x_2+x_3)+x_1 \cdot (x_2+x_3)+x_2 \cdot x_3)$$
$$K_4=C_4-C_5 \cdot (x_0+x_1+x_2+x_3)$$
$$K_5=C_5$$

VALOR DE Y PARA CUALQUIER VALOR DE X COMPRENDIDO ENTRE x_0 y x_4

$$y=K_5 \cdot x^4+K_4 \cdot x^3+K_3 \cdot x^2+K_2 \cdot x+K_1$$

NOTAS:

1) Le aconsejamos que realice el método para practicar sólo con tablas de una entrada y una salida, por ejemplo x=ángulo, y=seno(x).

2) La precisión del método depende del número de intervalos que se tomen, aumentando con el número de valores de x e y que se tomen. Los coeficientes que ha de elegir también dependen del número de valores. Así, tomando 3 valores para interpolar x_0, x_1, x_2 e y_0, y_1, y_2 habrá que calcular C_1, C_2, C_3 y K_1, K_2, K_3 y la ecuación final de y sería:

$$Y=K_3 \cdot x^2+K_2 \cdot x+K_1$$

```
10 REM Interpolacion por el metodo de Newton
20                '
30                '
40          Aproximacion de las funciones seno, coseno y tangente
50                '
60                '
80                '
90                '
110               '
120               '
130               '         El metodo de Newton aproxima una funcion mediante
140               '         un polinomio (de 4. grado en este caso), a partir
150               '         de cinco puntos equidistantes pertenecientes a la
160               '         funcion a interpolar; la funcion aproximada pasara
170               '         por dichos puntos
180 PRINT TAB(10); "Angulo"; TAB(20); "SEN(real)"; TAB(30); "SEN(CAL)";
    TAB(40); "COS(real)"; TAB(50); "COS(cal)"; TAB(60); "TANG(real)";
    TAB(70); "TANG(cal)"
190 FOR AN=0 TO 45
200 X=AN
210               '
220               '         Lee los 5 puntos equidistantes para el seno
230               '
240 READ X0, X1, X2, X3, X4, Y0, Y1, Y2, Y3, Y4
250 DATA 0, 22.5, 45, 67.5, 90, 0, .3826, .7071, .9238, 1
260               '
270 GOSUB 610     '         Acude a la subrutina para calcular los parametros
280               '         del polinomio de cuarto grado
290               '
300 S=Y
310               '
320               '          Lee los cinco puntos equidistantes para el coseno
330               '
340 READ X0, X1, X2, X3, X4, Y0, Y1, Y2, Y3, Y4
350 DATA 0, 22.5, 45, 67.5, 90, 1, .9238, .7071, .3826, 0
360               '
370 GOSUB 610     '         Acude a la subrutina para calcular los parametros
380               '         del polinomio de cuarto grado
390               '
400 C=Y
410               '
420               '         Lee los 5 puntos equidistantes para la tangente
430               '
440 READ X0, X1, X2, X3, X4, Y0, Y1, Y2, Y3, Y4
450 DATA 0, 15, 30, 45, 60, 0, .2674, .5774, 1, 1.73
460               '
470 GOSUB 610     '         Acude a la subrutina para calcular los parametros
480               '         del polinomio de cuarto grado
490               '
500 T=Y
510               '
520 RESTORE       '         Inicializa el puntero de los datas
530               '
540 PRINT TAB(10); AN; TAB(20); SIN(AN); TAB(30); S; TAB(40); COS(AN);
    TAB(50); C; TAB(60); TAN(AN); TAB(70); T
550 NEXT AN
560 GOTO 900
565               '         Subrutina
570               '         a partir de los 5 puntos calculo los cinco
580               '         parametros (k5, k4, k3, k2, k1) del polinomio
600               '
```

```
610 H=X1-X0
620 C1=Y0
630 C2=(Y1-Y0)/H
640 C3=(Y2-2*Y1+Y0)/(2*H*H)
650 C4=(Y3-3*Y2+3*Y1-Y0)/(6*H*H*H)
660 C5=(Y4-4*Y3+6*Y2-4*Y1+Y0)/(24*H*H*H*H)
670 K1=C1-X0*(C2-X1*(C3-X2*(C4-C5*X3)))
680 K2=C2-C3*(X0+X1)+(C4-C5*X3)*(X0*X1+X0*X2+X1*X2)-C5*X0*X1*X2
690 K3=C3-C4*(X0+X1+X2)+C5*(X0*(X1+X2+X3)+X1*(X2+X3)+X2*X3)
700 K4=C4-C5*(X0+X1+X2+X3)
710 K5=C5
720 Y=K5*X^4+K4*X^3+K3*X^2+K2*X+K1
730 RETURN
740 '
750 '
760 '
770 '
780 '
790 '
800 '
810 '          Valores de las funciones empleadas
820 '
830 '   Angulo-Seno            Angulo-Coseno           Angulo-Tangente
835 '
840 '   0.00     0.00          0.00     1.00           0.00     0.00
850 '   22.5     .3826         22.5     .9238          15       .2674
860 '   45       .7071         45       .7071          30       .5774
870 '   67.5     .9238         67.5     .3826          45       1.00
880 '   90       1.00          90       0.00           60       1.73
890 END
```

La tabla siguiente muestra los valores calculados (cal) por el programa y los deducidos (real) de unas tablas trigonométricas. SEN(real), COS(real) y TANG(real) son los valores deducidos de las tablas, y SEN(cal), COS(cal), TANG(cal) los valores calculados con el programa anterior.

Angulo	*Sen(real)*	*Sen(cal)*	*Cos(real)*	*Cos(cal)*	*Tang(real)*	*Tang(cal)*
0.0000	0.0000	0.0000	1.0000	1.0000	0.0000	0.0000
1.0000	.0175	.0174	.9998	.9999	.0175	.0161
2.0000	.0349	.0348	.9994	.9994	.0349	.0325
3.0000	.0523	.0521	.9986	.9987	.0524	.0492
4.0000	.0698	.0695	.9976	.9976	.0699	.0663
5.0000	.0872	.0869	.9962	.9963	.0875	.0836
6.0000	.1045	.1042	.9945	.9946	.1051	.1011
7.0000	.1219	.1216	.9925	.9926	.1228	.1189
8.0000	.1392	.1389	.9903	.9904	.1405	.1369
9.0000	.1564	.1561	.9877	.9878	.1584	.1551
10.0000	.1736	.1733	.9848	.9849	.1763	.1734
11.0000	.1908	.1905	.9816	.9817	.1944	.1919
12.0000	.2079	.2076	.9781	.9782	.2126	.2106
13.0000	.2250	.2247	.9744	.9744	.2309	.2294
14.0000	.2419	.2417	.9703	.9703	.2493	.2483
15.0000	.2588	.2586	.9659	.9659	.2679	.2674
16.0000	.2756	.2754	.9613	.9613	.2867	.2866
17.0000	.2924	.2922	.9563	.9563	.3057	.3060
18.0000	.3090	.3088	.9511	.9510	.3249	.3255
19.0000	.3256	.3254	.9455	.9455	.3443	.3452
20.0000	.3420	.3419	.9397	.9396	.3640	.3651
21.0000	.3584	.3583	.9336	.9335	.3839	.3852
22.0000	.3746	.3745	.9272	.9271	.4040	.4054
23.0000	.3907	.3907	.9205	.9204	.4245	.4259
24.0000	.4067	.4067	.9135	.9135	.4452	.4466
25.0000	.4226	.4226	.9063	.9062	.4663	.4676
26.0000	.4384	.4383	.8988	.8987	.4877	.4888
27.0000	.4540	.4540	.8910	.8909	.5095	.5104
28.0000	.4695	.4695	.8829	.8828	.5317	.5324
29.0000	.4848	.4848	.8746	.8745	.5543	.5547
30.0000	.5000	.5000	.8660	.8659	.5774	.5774
31.0000	.5150	.5151	.8572	.8571	.6009	.6006
32.0000	.5299	.5300	.8480	.8479	.6249	.6243
33.0000	.5446	.5447	.8387	.8386	.6494	.6485
34.0000	.5592	.5592	.8290	.8289	.6745	.6733
35.0000	.5736	.5736	.8192	.8191	.7002	.6987
36.0000	.5878	.5878	.8090	.8089	.7265	.7238
37.0000	.6018	.6019	.7986	.7986	.7536	.7517
38.0000	.6157	.6157	.7880	.7879	.7813	.7793
39.0000	.6293	.6294	.7771	.7771	.8098	.8078
40.0000	.6428	.6428	.7660	.7660	.8391	.8372
41.0000	.6561	.6561	.7547	.7547	.8693	.8675
42.0000	.6691	.6692	.7431	.7431	.9004	.8989
43.0000	.6820	.6820	.7314	.7313	.9325	9314
44.0000	.6947	.6947	.7193	.7193	.9657	.9651
45.0000	.7071	.7071	.7071	.7071	1.0000	1.0000

V

Características técnicas microcomputadoras

Discos, casetes, impresoras, sistemas operativos

Tabla V.1. Características fundamentales de microcomputadoras comerciales

Marca y modelo	*Microprocesador CPU*		*Memoria central*		*Memoria de masa*			*Software*		*Interfaces y observaciones*
					Casete	*Disco*				
	Modelo	*Bits*	*RAM (Kb/Mb)*	*ROM (Kb)*		*Flexible (K)*	*Rígido (M)*	*Sistemas operativos*	*Lenguajes*	
Amstrad CPC 464	Z-80A	8	64K	32K	Integrado	1×169(3")		AmsDOS-CP/M	Pascal, Fortran, BASIC, Logo	Centronics, usuario/disco
Amstrad CPC 6128	Z-80a	8	128K	64K	Opcional	1×169(3")		AmsDOS-CP/M	Pascal, Fortran, BASIC, Logo	Unidad de disco integrado Centronics
Amstrad PCW8256	Z-80A	8	256K	64K		1×180K		AmsDOS-CP/M	BASIC, Pascal, Fortran, Logo	Centronics, unidades de disco integradas
Amstrad PCW8512	Z-80A	8	5½K	64K		1×180K 1×720K		AmsDOS-CP/M	BASIC, Pascal, Fortran, Logo	Centronics, unidades de disco integradas
Apple IIc	65C02	8	128K	18K		1×143(5¼")		ProDOS, DOS 3.3	BASIC, Pascal	
Apple IIe	6502B		64-128K	16K		2×143(5¼")		ProDOS, DOS 3.3 UCSD, CP/M	BASIC, Pascal, Fortran, Cobol...	8 ranuras de expansión
Apple Macintosh	68000	16/32	128-512K	64K		1×400(3½")	20Mb	Finder	BASIC, Cobol, Fortran, Pascal, Logo	2×RS-232, 2 unidades de disco
Apple Macintosh Plus	68000	16/32	1-4M	128K		1×800(3½")	20Mb	Finder 5.0	BASIC, Cobol, Fortran, Pascal, Logo	
Apricot F1	8086	16	256K			1×315(3½")	10Mb	MS-DOS	BASIC, Cobol, Fortran, Pascal	RS-232, Paralelo
Apricot PC	8086	16	256-768K			2×711(3½")	10Mb	MS-DOS	BASIC, Cobol, Pascal, Fortran	RS-232, Centronics
Apricot portátil Xi	8086	16	256K	32K		1×711(3½")	10Mb	MS-DOS	BASIC, Cobol, Fortran, Pascal	RS-232, Centronics
Atari 130XE	6502C	8	64-128K	24K	Propio	1×128K(5¼")		Atari DOS	BASIC	Centronics
520ST	68000	16/32	512K	192K		1 o 2×349K(3½")		TOS-GEM	BASIC, Logo	RS-232, Centronics
1040ST	68000	16/32	1M	192K		1×720K(3½")	20M	TOS-GEM		RS-232, Centronics
Bull Micral 30 Micral 60	8088	15	128/768K	64K		1 o 2×360K(5¼")	10M	MS-DOS, Prologue, PICK, CP/M, OASIS	BASIC, Cobol, Logo, Pascal, Fortran	RS-232, Centronics
Burrough B25	80186	16	256/1000K	8K		2×630K(5¼")	10-40M	MS-DOS, CP/M-86	BASIC, Cobol, Fortran, Pascal, Logo	RS-232, Centronics
Canon A-200	8086	16	256K/512K	24K		2×360K(5¼")		MS-DOS	BASIC, Pascal, C, Fortran, Cobol	RS-232, Centronics
AS-300	80186	16	256K-512K	16K		1×720K(3½")	11,5/43M	MS-DOS		2 RS-232, 2 Paralelo
V-20	Z-80	8	64K	32K	Estándar	1×720K(3½")		MSX-DOS	BASIC	Cartuchos ROM, Paralelo
Casio FP-1000	Z-80A	8	64K	36K	Incorporado	2×320K(5¼")		CP/M	BASIC, Logo	1RS232, 1 Centronics, 1 IEEE 488
Casio FP-6000	8086	16	256K-768K	8K		2×320K(5¼")		MS-DOS	BASIC, Pascal, Cobol C y Logo	1RS-232, 1 Centronics

Tabla V.1. Características fundamentales de microcomputadoras comerciales. *(Continuación)*

Marca y modelo	Microprocesador CPU		Memoria central		Memoria de masa			Software		Interfaces y observaciones
					Casete	Disco				
	Modelo	Bits	RAM (Kb/Mb)	ROM (Kb)		Flexible (K)	Rígido (M)	Sistemas operativos	Lenguajes	
Columbia PC	8088	16	256K-1M	16K		2×320K(5¼")		MS-DOS, CP/M-86	BASIC, Pascal, Cobol, Fortran	2RS-232, 1 Centronics, 7 ranuras de expansión
Commodore 64(C-64)	6510	8	64K	16K	Propio	1×169K(5¼")		CBM	BASIC, Logo	Puerta usuario, Centronics
128(C-128)	6510/Z-80	8	128/512K	72K	Propio	1×169K(5¼")		CBM, CP/M	BASIC, Logo	Compatible C-64
PC10	8088	16	256/640K	8K		2×320K(5¼")		MS-DOS	GW BASIC, Pascal, Logo	5 ranuras
PC20	8088	16	256/640K	8K		1×320K(5¼")	10M	MS-DOS	GW BASIC, Pascal, Logo, etc.	5 ranuras, Centronics, RS-232
Amiga	68000	16	256/512K	256K		1×880K(3½")	1×20M	Amiga	BASIC, Logo, Pascal, C	8 M (RAM externos), discos 5¼" (opción)
Computec S/1-S/1 H1)	Z-80	8	64K	8K		2×320K(5¼")		CP/M	Pascal, BASIC...	1 RS-232
XP	8088	16	256-512K	8K		2×360K(5¼")	2×10M	MS-DOS	BASIC, Pascal...	Centronics, 8 ranuras
Compaq Deskpro 286	80286	16	512K	8K		1×1,2M(5¼")	1×20M	MS-DOS	BASIC, Pascal...	Disco de 70M.
Portátil 286										Ampliación 8M
Corona PC	8088	16	256-512K			1×360K(5¼")	1×10M	MS-DOS	BASIC, Pascal...	RS-232, Paralelo, 4 ranuras
Data General One	80C88	16	256K	32K		1×720K(3½")	Opción	MS-DOS CP/M-86	BASIC, Fortran...	Pantalla cristal líquido, portátil
Digital Rainbow	Z80A/8088	8 y 16	128-896K	24K		2×400K(5¼")	1×5, 10, 33M	MS-DOS, CP/M	BASIC, CBASIC, Fortran...	1RS-232 y 4 ranuras
Dragón 32/64/200	6809	8	64 K	16 K		1×175K(5¼")		05-9	BASIC, Pascal, Logo	Centronics, RS-232
Eina PC	8088	16	64-512 K			2×360K(5¼")		MS-DOS, CP/M-86	BASIC	2 RS-232, Centronics, 5 ranuras
Epsón PX-8	Z-80	8	64-128K	32K	Microcasete	1×360K(3½")		CP/M 2.2	BASIC, Pascal, Logo	1RS-232
QX-10	Z-80A	8	192-256K	2K		2×320(5¼")		CP/M	BASIC, Fortran, Cobol y Pascal	1RS-232, 1 Centronics, 5 ranuras
QX-16	Z80A y 8088	8 y 16	256-512K	6K		2×360K(5¼")		CP/M, MS-DOS	BASIC, Pascal...	1 RS-232, 1 Centronics, 3 ranuras
PC										
Ericson PC	8088	16	128-64	16K		1 ó 2×360K(5¼")	1×10 M	MS-DOS y CP/M	BASIC, Pascal...	6 ranuras
Portátil	8088	16	256-512	16K		2×360K(5¼")		MS-DOS y CP/M	BASIC, Pascal...	
Exelvisión EXL100	Z80	8	32K	32+8K	Estándar				BASIC	Enlace infrarrojos
Grid										
HP 110										
HP 150-II	8088									
HP-85	Propio	8	32K	48K	Cartucho	2×270K(3½")		HP	BASIC	Científico
HP-87	Propio	8	32K-544K	48K		2×270K(5¼")		HP, CP/M	BASIC	Científico
HP Vectra	80286	16	256-640K	64K		1×360K(5¼"), 1×1,2M	1×20 ó 1×40M	MS-DOS 3.1	BASIC, Pascal	7 ranuras, Pantalla tactil Reloj interno RS-232. Pantalla tactil

Tabla V.1. Características fundamentales de microcomputadoras comerciales. *(Continuación)*

Marca y modelo	*Microprocesador CPU*		*Memoria central*		*Memoria de masa*			*Software*		*Interfaces y observaciones*
					Casete	*Disco*				
	Modelo	*Bits*	*RAM (Kb/Mb)*	*ROM (Kb)*		*Flexible (K)*	*Rígido (M)*	*Sistemas operativos*	*Lenguajes*	
IBM PC	8088	16	64-640K	40K		2×360K(5¼")		MS-DOS	BASIC, Pascal...	5 ranuras
IBM Portátil	8088	16	256-512K	40K		2×360K(5¼")		MS-DOS	BASIC, Pascal...	8 ranuras
IBM XT	8088	16	128-640K	40K		1 ó 2×360K(5¼")	1×10M	MS-DOS	BASIC, Pascal...	8 ranuras
IBM AT	80286	16	256K-3M	64K		1×1,2M(5¼")	1×20M	MS-DOS	BASIC, Pascal...	8 ranuras infrarrojos
IBM IX	8088	16	64-512			1×720K(3½")				
ITT Xtra	8088	16	256-640K	32K		2×360K(5¼")		MS-DOS	BASIC, Pascal...	1RS-232, 1 Centronics, 5 ranuras
ITT Xtra XP	80286	16	512K-640K	32K		1 ó 2×360K(5¼")	1×10 ó 20M	MS-DOS	BASIC, Pascal...	1RS-232, 1 Centronics
Katson II	6502	8	48-64K	12K		2×143K(5¼")		DOS, CP/M	BASIC, Pascal...	8 ranuras
Kaypro 2/4/10	Z-80A	8	64K			2×320K(5¼")	1×10M	CP/M	BASIC, Pascal...	ranuras de expansión
2861(AT)	80286	16								
Multitech MPF III	6502	8	64K	24K		2×143K(5¼")		MPF DOS, CP/M	BASIC, Cobol	1 Centronics, 8 ranuras
Multitech MPF V-PC	8088	16	256K	8K		2×369K(5¼")	1×10M	MS-DOS	BASIC, Cobol...	1 RS232, 1 Centronics, 5 ranuras
NCR DMW	Z80A y 8088	8 y 16	128-512K	4K		2×320K(5¼")	1×10M	MS-DOS	BASIC, Fortran...	5 ranuras
NCR PC4i	8088	16	256-640K	16K		2×360K(5¼")	1×10M	MS-DOS	BASIC, Fortran...	7 ranuras
Olivetti M-10	Oki 80C85	8	24K	32K	Estándar			Propio	BASIC	1RS-232, 1 Centronics
M-20	Z8001	16	64-512K			1×320K(5¼")	10M	PCOS	BASIC	1RS-232, 4 ranuras
M-21	8086	16	256-640K	16K		2×360K(5¼")	10M	MS-DOS	BASIC, Pascal...	1RS-232, 1 Centronics
M-24	8086	16	256-640K	16K		2×360/720K(5¼")	10M	MS-DOS	BASIC, Pascal...	7 ranuras
Sanyo MBC 550-2	8088	16	128-256	8K		2×360K(5¼")		MS-DOS	BASIC, Pascal...	
MPC 100 (MSX)	Z-80A	8	64K	32K	Estándar	1×360K(5¼")		MSX-DOS	BASIC	Opción disco rígido
MBO 675										
Secoinsa FM-7	68B09	8	117K	32K		2×304K(5¼")	1×10	CP/M	BASIC, Logo, Pascal	
FM-100	y Z-80A									
Sharp MZ-5600	8086	16	256-512K	16K		2×790K(5¼")	1×10	MS-DOS	BASIC	2RS-232, 1 Centronics
PC-5000										
Sinclair ZX Spectrum	Z-80A	8	48K	16K	Estándar	Opcional (3½")			BASIC, Pascal, Logo	Cartuchos de cinta
Spectrum 128	Z-80A	8	128K	32K	Estándar	Opcional (5¼")			BASIC, Pascal, Logo	Cartuchos de cinta
QL	68008	16	128K	32K	Cartuchos	Opcionales		QDOS	BASIC	2RS-232
Sony Hit-Bit 75	Z-80A	8	64	32K	Cartuchos	2×360K(3½")		MSX-DOS	BASIC, Logo	MSX
Hit-Bit F500P	Z-80A	8	64K	48K	Cartuchos	1×120K(3½")			BASIC MSX-2	MSX-2, 128K RAM pantalla
Spectravideo 728	Z-80A	8	80-128K	32K	Propio	2×320(5¼")		CP/M2.2/MSX-DOS	BASIC, Pascal...	MSX
Sperry PC/HT	8088	16	256-640K	24K		2×360K(5¼")	1×20M	MS-DOS	BASIC, Pascal...	RS-232, 6 ranuras
PC/IT	80286	16	512K			1×360K(5¼") 1×1.2Mb	1×44M	MS-DOS, Xenix	BASIC, Pascal	Computable AT

Tabla V.1. Características fundamentales de microcomputadoras comerciales. *(Continuación)*

Marca y modelo	*Microprocesador CPU*		*Memoria central*		*Memoria de masa*			*Software*		*Interfaces y observaciones*
					Casete	*Disco*				
	Modelo	*Bits*	*RAM (Kb/Mb)*	*ROM (Kb)*		*Flexible (K)*	*Rígido (M)*	*Sistemas operativos*	*Lenguajes*	
Tandy 200	80C85	8	24-72K	72K	Estándar	1×320K(5¼")		MS-DOS	BASIC, Pascal...	
Tandy-3000	80286		512K-12M			1×1,2M(5¼")				
Televideo PC	8088	16	256K-640K	24K		2×360K(5¼")		MS-DOS	BASIC, Pascal...	
AT	80286	16	256-640K	32K		1×1,2M(5¼")		MS-DOS	BASIC, Pascal...	
Texas TI-PC	8088	16	256-768K	8K		2×360K(5¼")	1×10	MS-DOS	BASIC, Pascal...	Reconocedor de voz
Business Pro	80286	16	256-640K	32K		1×1,2M(5¼")	1×20	MS-DOS	BASIC, Pascal...	
Toshiba HX-10E										
T-350	8086	16	256-512K			2×1.2Mb	10 Mb ó 20 Mb	OASIS-16	BASIC, Pascal	RS-232, Centronics
T1100	8086	16	256-512K			1×720Kb		MS-DOS	BASIC, Pascal...	Discos 3½"/pila interna
T1500	8086	16	128-640K			1×360K(5¼")	1×10M	MS-DOS//CP/M-86		
2100	8086	16	256-640K			1×360K(5¼")	Opción	MS-DOS		Portátil
3100	80286	16	640K-2.64M			1×360K(5¼")	Opción	MS-DOS		Portátil
Víctor PC	8088	16	128-896K			2×360K(5¼")	1×10M	MS-DOS	BASIC, Pascal...	
Wang PC	8086	16	128-640K			2×360K(5¼")	1×10M	MS-DOS	BASIC, Pascal...	
Zenith Z-150 PC	8088	16	128-640K			2×360K(5¼")	1×10M	MS-DOS	Pascal, BASIC...	
Z-2000										
Philips VG8010	Z-80	8	64K(16K vídeo)	32K	Estándar	1×360K(3½")				Centronics
VG8020	Z-80	8	80K(16K vídeo) 192K	32K		1×360K(3½")		MSX-DOS	BASIC, Logo	MSX
Thomson TO-9	6508	8	128K-192K	128K		(3½")				

Tabla V.2. Impresoras matriciales.

Marca	*Modelo*	*Velocidad impresión (CPS)*	*Anchura de carro*	*Bidireccional (S/N)*	*Matriz de puntos*	*Interfaz RS-232(S) Centronics (P)*	*Gráficos (G) Color (C)*	*Observaciones*
Admate	DP-80	100	80	S	9×9	S	S	
	DP-130 I	130	132	S	7×8	P	S	
Apple	Imagewriter	120	132			P	S	
BMC	BX 80	80		S	8×9	S, P		
	BX 100	100		S	8×9	S, P		
Brother	2024	160 cps	136	S	18×9	S, P	G	Tracción, Fricción
Bull	Prt 7220	200		S	9×9	S, P	G	Tracción, fricción, alimentador de papel
Canón	PW-1080 A	160	136	S	11×9	P	Gl	
	A-1200	120	136	S	9×7	P	G	
	1210	40	40/80	S			G, C	Tipo Ink-jet
	PW-1156A	160	136	S	11×9	P	G	
Centronics	154	120	132	S			G	
	150	150	132	S		S, P	G	
	2141-5/5	160	80	S	11×19		G	
	2142/5/6/7	200	132	S	11×19		G	
Citizen	MSP-10/15	160/40	80/136	S	9×9	S, P	G	
	MSP-20/25	200/50	80/136	S	9×9	S, P	G	
	120-D	120/25	80/136	S	9×9	S, P	G	
C. Itoh	7500	105	80	S	9×7	P	G	
	8510BP	120	80	S	8×8	P	G	
	8510EP	180	80	S	8×8	P	G	
	1550SP	120	132	S	8×8	P	G	
	1570	200/50	132	S	18×18	P	G	
	CI 3500-10/20	350/87	136	S				
Commodore	MDS 1361	160	132	S	7×6	P	G	
Epson	FX-80/FX-85	160 cps	80	S	9×9	S, P		Tracción, fricción
		160 cps	132	S	11×9			Tracción, fricción
	FX-100/FX-105							
	LQ 1500	200	136			S, P		
	LX-80 F/T	100	80			S, P		
	LQ-800/LQ-1000	180 cps	80/132	S	9×23	S, P	G	60 cps (NLQ), Tracción, fricción, alimentador de papel
Facit	4510	120	80	S	9×9/9×15	S, P	G	Tracción, fricción
	4511	160	132	S	9×9/18×17	S, P	G	Tracción, fricción
Honeywell	36CQ	300		S	7×9	S, P	G	Tracción, fricción, alimentador de papel
IBM	Proprinter	200	80	S		S, P	G	Alta calidad letra (40 cps)
Juki	5510/5520	180		S	9×9	S, P	G, C	Tracción, fricción, alimentador de papel
Nec	Pinwriter P2	180	137	S	11×7	S, P	G	Tracción, fricción, alimentador de papel
	Pinwriter P3	180	136	S	11×7	S, P		Tracción, fricción, alimentador de papel
	Pinwriter P5	264	136	S	17×9	S, P	G	Tracción, fricción, alimentador de papel
Oki	Microline 82A	120		S	9×9	S, P	G	Tracción, fricción, alimentador de papel
	83A	120		S	9×9			

Tabla V.2. Impresoras matriciales. *(Continuación)*

Marca	*Modelo*	*Velocidad impresión (CPS)*	*Anchura de carro*	*Bidireccional (S/N)*	*Matriz de puntos*	*Interfaz RS-232(S) Centronics (P)*	*Gráficos (G) Color (C)*	*Observaciones*
	92P	160		S	9×9	P	G	Tracción, fricción, alimentador de papel
	93P	160		S	9×9	P	G	Tracción, fricción, alimentador de papel
Riteman	R10	120 cps		S	9×9	S, P	G	Tracción, fricción, alimentador de papel
	C+, F+	105 cps		S	9×9	S (F+, propio)	G	Tracción, fricción, alimentador de papel
	RI5	160 cps		S	9×9	S, P	G	Tracción, fricción
Seikosha	GP-700	50	80	N	7×8	P	G, C	Tracción, fricción
	GP-1000							
	GP 1000 AS							
	GP 1300 AI							
	BP S200							
	BP 5420							
Star	SD 10/15	160	80/132	S	9×11	S, P	G	Tracción, fricción, alimentador de papel
	SG-10/15	120	80/132		9×11	S, P	G	Tracción, fricción, alimentador de papel
	SR-10/15	200	80/132		9×11	S, P	G	Tracción, fricción, alimentador de papel
Texas	11850	150	80		9×9	S, P		Tracción, fricción

Tabla V.3. Impresoras láser.

Marca	*Modelo*	*Velocidad de impresión*	*Resolución gráfica (puntos/pulgada²)*	*Capacidad Buffer*	*Interfaces Serie, S Paralelo, P*	*Observaciones*
Apple	Laser Writer	12 hpm	300×300		S	µp 68000, 500K(ROM), 1500K(RAM)
Canon	LBP-CX	8 hpm	300×300	1.2 M	S, P	
HP	Laserjet	8 hpm	300×300	59 K	S	
Wang	LPS-12	12 hpm	300×309	64 K	Propio	
Xerox	2700	12 hpm	300×300		S, P	
	4045	10 hpm	300×300			
Facit	Opus 1	12 hpm	300×300			

Tabla V.4. Impresoras de inyección.

Marca	*Modelo*	*Velocidad (cps)*	*Matriz*	*Ancho carro*	*Gráficos/Color*	*Interfaces*
Canon	A-1210	40	5×7	80	G, C	Centronics
HP	Thinkjet	150	11×12	80	G	Serie, paralelo
Nixdorf	Inkjet	210	13×9	80	G	

VI

Guías de referencia rápida

- — IBM/PC
- — APPLE II
- — Hewlett Packard HP-85, 86, 87
- — AMSTRAD CPC 464/CPC 664/6128
- — MSX
- — ZX Spectrum/+/128K
- — COMMODORE 64/128
- — DRAGON-32/64/200 (MBASIC)

IBM/PC

Teclado

Tecla ENTER (equivalente a RETURN).

Teclas ALT y CTRL: Se utilizan en unión con otras teclas para enviar órdenes o palabras clave a la computadora. Por ejemplo, al presionar la tecla ALT y escribir P se generará la palabra PRINT en la pantalla.

Teclado numérico: Conjunto de teclas a la derecha del teclado estándar.

Teclas de función: Son cinco filas de dos teclas en el lado izquierdo del teclado rotuladas de F1 a F10. El objeto de las teclas de función es facilitar la introducción de órdenes utilizadas frecuentemente, de modo que es igual pulsar la tecla de función que escribir su nombre por teclado.

```
F1    LIST
F2    RUN"
F3    LOAD"
F4    SAVE"
F5    CONT
F6    ,"LPT1;"
F7    TRON
F8    TROFF
F9    KEY
F10   SCREEN 0, 0, 0
```

Funciones de control del sistema

RESET (Reinicialización)
Se activa al pulsar simultáneamente las teclas CTRL, ALT y DEL. Una reinicialización del sistema origina que la computadora personal se coloque en la misma condición que si se acabase de encender.

BREAK
La función BREAK *(interrupción, ruptura)* se activa al pulsar simultáneamente las teclas CTRL y SCROLL LOCK/BREAK. Esta función abortará o interrumpirá cualquier programa que se esté ejecutando en la computadora.

PAUSE
La función PAUSE *(pausa)* se produce al pulsar simultáneamente CTRL y NUM LOCK. Esta función detiene temporalmente el programa en memoria. Pulsando cualquier tecla después de la pausa el programa se reanuda de nuevo.

Teclas de edición simples

Tecla BACKSPACE: Al pulsar esta tecla se borra o elimina el carácter a la izquierda del cursor y se desplaza éste una posición a la izquierda.

Tecla ESC: La tecla ESC (Escape) elimina líneas de la pantalla.

Teclas de movimiento de cursor

Las teclas numéricas 8, 2, 4 y 6 permiten al presionarlas desplazar el cursor una posición arriba, abajo, izquierda o derecha.

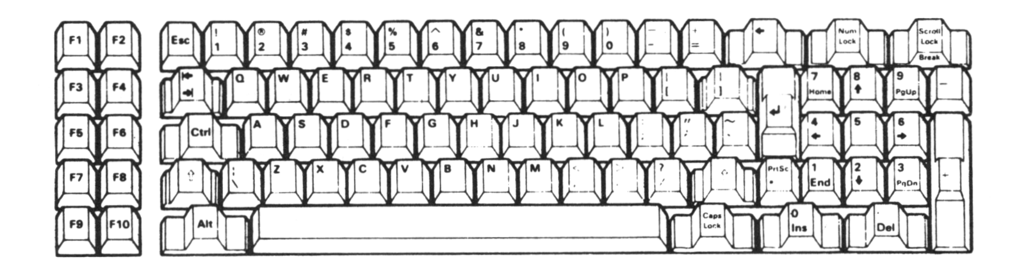
Movimiento del cursor por palabras: Combinando la tecla CTRL con las teclas CURSOR LEFT (4) y CURSOR RIGHT (6) puede mover el cursor una palabra a la izquierda o a la derecha cada vez.

Tecla END: La tecla END desplaza el cursor al final del último carácter de la línea que se está escribiendo.

CTRL END: Cuando se presiona la tecla CTRL y se pulsa la tecla END, la parte de la línea que está a la derecha del cursor se borra o elimina.

Tecla HOME: La tecla TAB desplaza el cursor a posiciones determinadas de tabulación (1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 80).

Tecla DEL: Se elimina un carácter cada vez que se pulsa la tecla DEL.



Teclado BASIC

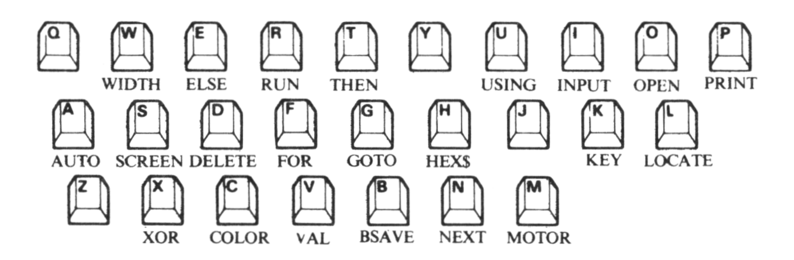
Ctrl Alt Del = Reinicialización del sistema.

Esc (Escape) Elimina líneas de pantalla.

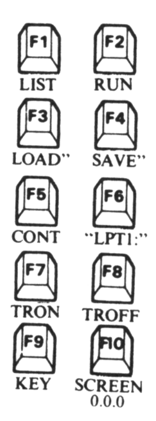
Ctrl Num Lock Ralentiza el programa.

Ctrl Break Aborta el programa, vuelve a petición BASIC

Introducir palabras claves pulsando Alt + tecla de letra



Teclas de función predefinidas



Introducir funciones de carácter con tecla numérica





Borrar pantalla






Palabra a la izquierda

Palabra a la derecha






Elimina un carácter

Inserta un espacio

Borra hasta el final de la línea

Tecla cursor con Ctrl tecla:


Borrar pantalla


Cursor hacia la izquierda una palabra


Cursor hacia la derecha una palabra


Borrar hasta fin de la línea

Tecla cursor con  **tecla:**

Introducir caracteres ASCII de 0 a 255 con Alt

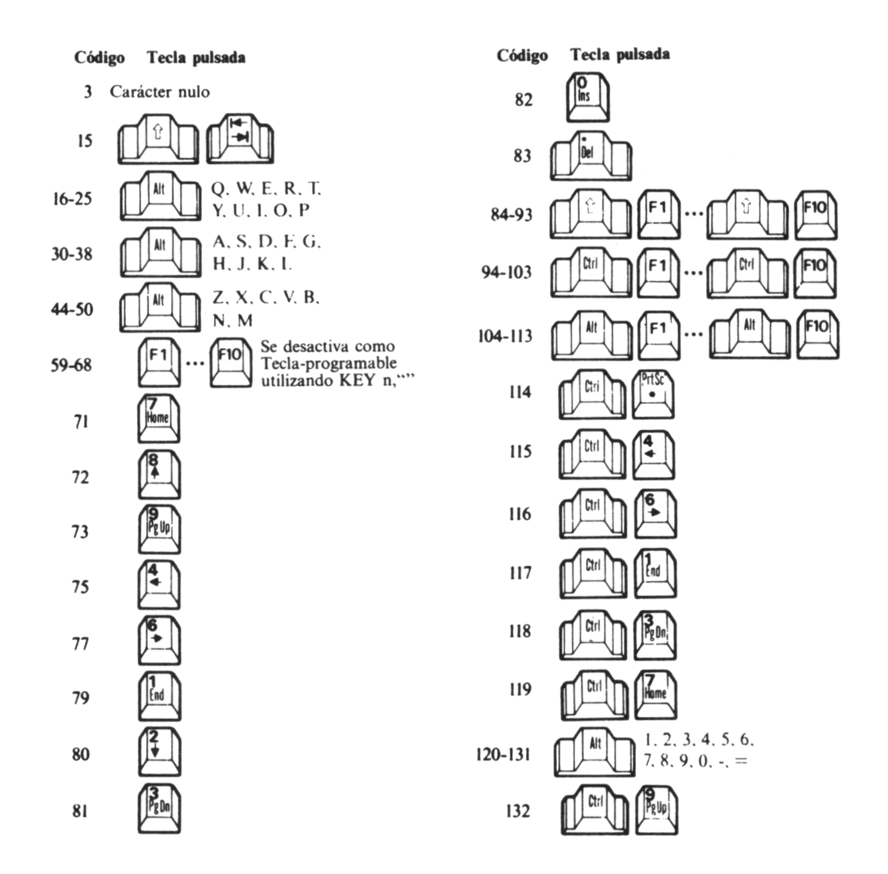
Ejemplo:





Imprime "E" (ASCII 69)

Códigos de teclas ampliadas. Códigos de dos octetos devuelto por INKEY$
A$=INKEY$: IF LEN(A$)=2 THEN CODE=ASC(RIGHT$(A$,1))



PALABRAS CLAVES CON LA TECLA ALT

Asignación de palabras con la tecla ALT

ALT	*Palabra clave*	*ALT*	*Palabra clave*	*ALT*	*Palabra clave*
A	AUTO	J	Ninguna	S	SCREEN
B	BSAVE	K	KEY	T	THEN
C	COLOR	L	LOCATE	U	USING
D	DELETE	M	MOTOR	V	VAL
E	ELSE	N	NEXT	W	WIDTH
F	FOR	O	OPEN	X	XOR
G	GOTO	P	PRINT	Y	Ninguna
H	HEX$	Q	Ninguna	Z	Ninguna
I	INPUT	R	RUN		

Tabla VI.1. Caracteres especiales BASIC.

Carácter	*Nombre*	*Carácter*	*Nombre*
	Espacio en blanco	!	Punto de exclamación
=	Signo igual o símbolo asignación	&	Ampersand (*y* comercial)
+	Signo más o símbolo concatenación	,	Coma
−	Signo menos	.	Punto decimal
*	Símbolo asterisco o multiplicación	’	Apóstrofo
/	Símbolo barra inclinada o división	;	Punto y coma
\	Símbolo división entera	:	Dos puntos
∧	Símbolo circunflejo o exponenciación	?	Signo de interrogación
(	Paréntesis a la izquierda	<	Menor que
)	Paréntesis a la derecha	>	Mayor que
%	Tanto por ciento	”	Comillas
#	Signo número	—	Guión-Subrayado
$	Signo dólar		

Tabla VI.2. Variables constantes.

Tipo	*Dígitos*	*Octetos*	*Rango de valores*	*Ejemplo*	*Nombre de la variable*
Entero	5	2	−32768 a 32767		
Simple precisión	7+2 exp.	4	desbordamiento=1.7E38	1.23456−7E−12	V3 o NNN
Doble precisión	16+2 exp.	8	desbordamiento=295E−39 negativo	1.23456−789D32	B12#
Hexadecimal	4	2	0000 a FFFF	& H94EF	
Octal	6	2	0000000 a 177777	& 03070&307	
Cadena		1−255		«hora=3:40»	PATAL$

A% = Entero
S = Simple precisión
D# = Doble precisión

Tabla VI.3. Operadores.

Operadores aritméticos			*Operadores relacionales*		
Operador	*Función*	*Ejemplo*	*Operador*	*Función*	*Ejemplo*
^ , **	Exponenciación	2^4=16	=	Igual	4=5 falso (0)
*	Multiplicación	7*5=35	<>	Distinto	4<>5 cierto (−1)
/	División	10/4=25	<	Menor que	4<5 cierto
\	División entera	10\6=1	>	Mayor que	4>5 falso
MOD	Módulo	10 MOD 6=4	<=	Menor o igual que	4<=5 cierto
+	Suma	4+7=11	>=	Mayor o igual que	4>=5 falso
−	Resta	8−2=6			

Tabla VI.4. Operdores lógicos.

Operador	*Función lógica*
NOT	Complemento lógico
AND	Intersección «Y»
OR	Unión «0»
XOR	OR exclusiva
IMP	Implicación
EQV	Equivalencia

Tabla VI.5. Operaciones lógicas.

Operador	*Tabla de verdad*		
NOT	**X**		**NOT X**
	V		F
	F		V
AND	**X**	**Y**	**X AND Y**
	V	V	V
	V	F	F
	F	V	F
	F	F	F
OR	**X**	**Y**	**X OR Y**
	V	V	V
	V	F	V
	F	V	V
	F	F	F
XOR	**X**	**Y**	**X XOR Y**
	V	V	F
	V	F	V
	F	V	V
	F	F	F
IMP	**X**	**Y**	**X IMP Y**
	V	V	V
	V	F	F
	F	V	V
	F	F	V
EQV	**X**	**Y**	**X EQV Y**
	V	V	V
	V	F	F
	F	V	F
	F	V	V

Ordenes (comandos) del sistema

AUTO
Genera números de línea automáticamente después que ha pulsado la tecla ENTER.

CLEAR
Inicializa las variables numéricas a cero y las variables de cadena nulas y activas opcionalmente al final de la memoria y el espacio de la pila.

CONT
Reanuda la ejecución del programa después de una interrupción.

DELETE
Borra las líneas especificadas de memoria.

EDIT
Visualiza una línea para la edición.

LEFT$(X$, N)
Devuelve los N caracteres más a la izquierda de X$.

LIST
Lista el programa actual.

LLIST
Imprime total o parcialmente la parte especificada del programa actualmente en memoria en la impresora.

NEW
Borra el programa actualmente en memoria.

RENUM
Renumera las líneas de un programa.

RUN
Comienza la ejecución del programa actualmente en memoria o carga y ejecuta un programa por su nombre.

STOP
Termina la ejecución del programa.

SYSTEM
Salida de BASIC y retorno al sistema operativo DOS.

TRON
Activa la ejecución del programa paso a paso.

TROFF
Desactiva la ejecución del programa paso a paso.

MKS$(X)
Convierte la expresión de precisión simple en una cadena.

MKD$(X)
Convierte la expresión de precisión doble en una cadena.

OCT$(N)
Convierte N en una cadena octal.

RIGHT$(X$, N)
Devuelve los N caracteres más a la derecha de X$.

SPACE$(N)
Devuelve una cadena de N espacios.

SPC(N)
Imprime N espacios en una sentencia PRINT o LPRINT.

STR$(X)
Convierte X en un valor de cadena.

STRING$(N,X$)
Se repite el primer carácter de X$, N veces.

STRING$(N,M)
El carácter en código ASCII de valor M se repite N veces.

SWAP A$, F$
Intercambia los contenidos de dos cadenas (no necesitan ser de la misma longitud).

TIME$
Devuelve la hora.

VAL(X$)
Convierte una cadena en valor numérico.

VARPTR$(V)
Devuelve una cadena de tres octetos que contiene el tipo de variable y la dirección de la variable en memoria.

Funciones trigonométricas derivadas

DEF FNSEC(X)=1/COS(X)
Secante

DEF FNCSC(X)=1/SIN(X)
Cosecante

DEF FNCOT(X)=1/TAN(X)
Cotangente

DEF FNARCSIN(X)=ATN(X/SQR(1-X*X))
Arco seno

DEF FNARCCOS(X)=1.570796-ATN(X/SQR(1-X*X))
Arco coseno

DEF FNARCSEC(X)=ATN(SQR(X*X-1))+(X<0)*3.141593
Arco cosecante

DEF FNARCCSC(X)=ATN(1/SQR(X*X-1))+X<0)*3.141593
Arco cotangente

DEF FNARCCOT(X)=1.570796-ATN(X)
Arco tangente

DEF FNSINH(X)=(EXP(X)-EXP(X))/2
Seno hiperbólico

DEF FNCOSH(X)=(EXP(X)+EXP(-X))/2
Coseno hiperbólico

DEF FNTANH(X)=(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
Tangente hiperbólica

DEF FNARCSINH(X)=LOG(X+SQR(X*X+1))
Arco seno hiperbólico

DEF FNARCCOSH(H)=LOG(X+SQR(X*X-1))
Arco coseno hiperbólico

DEF FNARCTANH(H)=LOG((1+X)/(1-X))/2
Arco tangente hiperbólico

DEF FNARCSECH(X)=LOG((1+SQR(1−X*X))/X)
Arco cosecante hiperbólico
DEF FNARCCSH(X)=LOG((1+SGN(X)*SQR(1+X*X))/X)
Arco cotangente hiperbólico
DEF FNARCCOTH(X)=LOG((X+1)/(X−1))/2

Funciones numéricas

MATEMATICAS *(ángulos en radianes)*

ABS(X)
Proporciona el valor absoluto de X.

ATN(X)
Arco tangente de X.

CDBL(X)
Convierte X a un número de doble precisión.

CINT(X)
Convierte X a un entero, redondeando.

COS(X)
Coseno de X.

CSNG(X)
Convierte X a un número de simple precisión.

EXP(X)
Eleva *e* a la potencia X.

FIX(X)
Trunca X a un entero.

INT(X)
El entero mayor que sea menor o igual que X.

LOG(X)
Logaritmo natural (base *e*) de X.

RND(X)
Número aleatorio.

SGN(X)
Signo de X.

SIN(X)
Seno de X.

SQR(X)
Raíz cuadrada de X.

TAN(X)
Tangente de X.

De E/S, relacionadas con cadenas Y especiales

CSRLIN
Proporciona la posición vertical de la línea del cursor.

CVI(X$)
Convierte X$ a un número entero.

CVS(X$)
Convierte X$ a un número de simple precisión.

CVD(X$)
Convierte X$ a un número de doble precisión.

ERL
Proporciona el número de línea donde ocurrió el último error.

ERR
Proporciona el número del código de error del último error.

ERROR
Permite la simulación de un error BASIC o que se definan los códigos de error.

FRE(X$)
Proporciona la cantidad de espacio libre de memoria no utilizada.

INP(N)
Lee un octeto del *port* n.

LPOS(N)
Proporciona la posición del carro de la impresora.

PEEK
Lee el octeto en la posición de memoria n.

EN(N)
Lee el lápiz óptico.

POINT(X, Y)
Proporciona el color del punto (X, Y) (modo gráfico).

POS(N)
Proporciona la posición de la columna del cursor.

SCREEN(X, Y, Z)
Proporciona el carácter o color en la posición (X, Y).

STICK(N)
Proporciona las coordenadas del controlador de juegos.

SWAP A, B
Intercambia los valores de dos variables numéricas.

TAB(N)
Tabula a la posición N en una sentencia PRINT o LPRINT.

USR(X)
Llama a una subrutina en lenguaje máquina con argumento X.

VARPTR *(variable)*
Devuelve la dirección de la variable en memoria.

Funciones de cadena

ASC(X$)
Proporciona el código ASCII del primer carácter de la cadena.

CHR$(N)
Proporciona el carácter correspondiente al código ASCII de N.

DATE $
Proporciona la fecha.

HEX $
Convierta N a una cadena hexadecimal.

INKEY $
Lee un carácter del teclado.

INSTR(N, X$, Y$)
Posición de la primera aparición de Y$ en X$.

LEN(X$)
Longitud de X$.

MID$(X$, N, M)
Proporciona M caracteres de X$ comenzando en la posición N.

MKI$(X)
Convierte la expresión entera especificada en una cadena.

Para convertir grados sexadecimales a radianes utilizar las fórmulas siguientes

GRADOS=X*180/3.141592 (X en radianes).

RADIANES=X*3.141592/180 (X en grados).

Sentencias

BEEP
Genera un pitido en el altavoz.

CALL
Llamada a una subrutina en lenguaje máquina.

CHAIN
Transfiere el control a otro programa y pasa a él las variables desde el programa actual.

CLEAR
Pone todas las variables numéricas a cero y las variables de cadena las hace nulas.

CLOSE
Cierra un dispositivo o fichero de E/S.

CLS
Borra la pantalla.

DATA
Crea una lista de constantes numéricas y de cadenas que se asignan a variables a través del uso de la sentencia READ.

DEF EN
Define y nombra una función que describe el usuario.

DEF SEG
Define el segmento actual de almacenamiento.

DEF TYPE
Declara los tipos de variables como entero, simple precisión, doble precisión o cadena.

DEF USR
Especifica la dirección de comienzo de una subrutina en lenguaje máquina, que es llamada posteriormente por la función USR.

DIM
Especifica los valores máximos de las variables subcritas de las matrices (arrays) y asigna el correspondiente espacio de memoria.

END
Termina la ejecución de un programa.

EOF
Indica una condición de fin de fichero.

ERASE
Elimina las matrices de un programa para permitir el redimensionado.

ERR y ERL
Devuelve el código del error y el número de línea asociado con un error.

ERROR
Simula la aparición de un error BASIC o permite definir sus propios códigos de error.

FOR-NEXT
Ejecuta la serie de instrucciones interiores al lazo un número dado de veces.

GOSUB
Bifurca a una subrutina.

GOTO
Bifurca incondicionalmente al número de línea indicado.

IF-THEN-ELSE
Si la expresión después de IF es verdad, se ejecuta(n) la(s) sentencia(s) entre THEN y ELSE; si no es verdad, se ejecuta(n) la (s) sentencias después de ELSE.

IF-GOTO-ELSE
Igual que la anterior.

INPUT
Recibe entrada desde el teclado durante la ejecución del programa.

KEY
Activa o visualiza las teclas programables.

LET
Asigna el valor de una expresión a una variable.

LINE INPUT
Leer una línea completa (hasta 254 caracteres) desde el teclado en una variable de cadena, ignorando delimitadores.

LPRINT
Imprime datos en impresora.

LPRINT USING
Imprime datos en impresora.

MOTOR
Conecta y desconecta el casete desde programa.

ON ERROR
Activa la localización de errores y especifica la primera línea de la subrutina de tratamiento de errores.

ON GOSUB
Bifurca a una de varias subrutinas de acuerdo con el valor de una expresión.

ON GOTO
Bifurca a una de varias líneas de acuerdo con el valor de la expresión.

ON KEY
Bifurca a un número de línea o subrutina cuando una tecla de función o de control de cursor se pulsa.

OPTION BASE
Declara el valor mínimo de los subíndices o subscritos de una matriz.

OUT
Envía un octeto a un port de salida de la máquina.

PLAY
Suena música de acuerdo con una cadena.

PRINT
Visualiza datos en la pantalla.

PRINT USING
Imprime cadenas o números utilizando un formato especificado.

RANDOMIZE
Reinicia la semilla generadora de números aleatorios.

READ
Lee los valores de una sentencia DATA y los asigna a variables.

REM
Inserta comentarios explicativos en un programa.

RESTORE
Permite que las sentencias DATA sean leídas desde una línea especifica.

RESUME
Continúa la ejecución de un programa después que se ha ejecutado un programa de recuperación de errores.

RETURN
Retorna al programa principal después de la ejecución de la subrutina.

RND
Devuelve un número aleatorio entre 0 y 1.

SOUND
Genera sonidos a través del altavoz.

SYSTEM
Salida de BASIC y retorno a DOS.

USR
Llama a la subrutina en lenguaje máquina indicada por el argumento.

WAIT
Suspende la ejecución del programa mientras se supervisa el estado del port de entrada de la máquina.

WHILE y WEND
Ejecuta una serie de sentencias en un bucle mientras una condición dada es verdadera.

WIDTH
Establece la anchura de las líneas de salida con un determinado número de caracteres.

WRITE
Saca datos en la pantalla.

APPLE II

Variables simples

Tipo	*Nombre*	*Rango*
Numérica	AB	+/−9.999999 E+37
Entera	AB %	+/−32767
Cadena	AB$	0 a 255 caracteres

A es una letra, B es una letra o dígito. El nombre puede ser de más de dos caracteres, pero sólo los dos primeros son significativos: AB y ABCD son la misma variable.

Variables suscritas (con subíndice)

Tipo	*Nombre del elemento de la matriz (array)*
Real	AB (3, 12, 7)
Entera	AB % (3, 12, 7)
Cadena	AB$ (3, 12, 7)

El tamaño de la matriz está limitada por la memoria disponible.

Ordenes de edición

→ Desplaza el cursor hacia adelante a lo largo de una línea de visualización. Cada carácter sobre el que se pasa se copia a memoria como si se hubiera tecleado. No altera la pantalla de visualización.

← Desplaza el cursor hacia atrás a lo largo de una línea de visualización, borrando de memoria, pero no de la pantalla de visualización, todos los caracteres sobre los que pasa.

REPT Hace que el carácter impreso por otra tecla se repita mientras las dos teclas se mantengan pulsadas. La tecla REPT se debe pulsar *después* de la otra tecla.

CTRL-X El APPLE II descarta la línea de visualización actual y desplaza el cursor al margen izquierdo de la siguiente línea inferior.

Secuencias de la tecla Esc

Los siete comandos de edición siguientes son secuencias de dos teclas. En cada caso, pulse la tecla Esc, libérela y luego pulse la segunda tecla de la secuencia.

ESC-A Desplaza el cursor una posición hacia la derecha. No altera la pantalla de visualización ni cambia la memoria.

Esc-B Desplaza el cursor una posición hacia la izquierda. No altera la pantalla de visualización ni cambia la memoria.

Esc-D Desplaza el cursor una línea hacia arriba. No altera la pantalla de visualización ni cambia la memoria.

Esc-E Elimina todos los caracteres desde el cursor hasta el final de la línea de visualización.

Esc-F Elimina todos los caracteres desde el cursor hasta el final de la pantalla de visualización.

Esc-@ Borra la pantalla y desplaza el cursor a la esquina superior izquierda.

Ordenes en modo editor

Los cuatro comandos siguientes requieren el Monitor de Autolanzamiento. Solamente son efectivos en modo *editor*. Introduzca el modo editor pulsando la tecla Esc y abandónelo, pulsando cualquier tecla que no sea la I, J, K, M, Rept, Eeturn o Shift.

I Desplaza el cursor una línea hacia arriba sin dejar el modo editor.

J Desplaza el cursor una posición hacia la izquierda sin dejar el modo editor.

K Desplaza el cursor una posición a la derecha sin dejar el modo editor.

M Desplaza el cursor una línea hacia abajo sin dejar el modo editor.

Tabla VI.6. Operadores.

	Prioridad	*Operador de Integer BASIC*	*Operador de Applesoft*	*Significado*
	Superior 9	()	()	El paréntesis indica el orden de evaluación
Operadores aritméticos	8	^	^	Potenciación
	7	–	–	Signo menos
	6	*	*	Multiplicación
	6	/	/	División
	6	MOD	No disponible	Resto de la división**
	5	+	+	Suma
	5	–	–	Resta
Operadores de relación	4	=	=	Igual que
	4	#	< > ó > <	Distinto a
	4	<	<	Menor que
	4	>	>	Mayor que
	4	<=	<= ó =<	Menor que o igual
	4	>=	>= ó =>	Mayor que o igual
Operadores booleanos	3	NOT	NOT	Complemento lógico
	2	AND	AND	AND lógico
	1 Inferior	OR	OR	OR lógico

** Sólo para Integer BASIC.

Tabla VI.7. Operadores de relación.

Operador de Integer BASIC	*Operación*	*Operador de Applesoft*
<	Menor que*	<
>	Mayor que*	>
=	Igual a	=
# o < >	No igual a	< > o > <
>=	Mayor que o igual a*	>=o=>
<=	Menor que o igual a*	<=o=<

* No permitidos con cadenas en Integer BASIC.

ORDENES (COMANDOS)

Del sistema y utilidades

CTRL-C
Se utiliza en modo inmediato para detener el listado o el programa.

CTRL-S
Ralentiza la presentación del listado en pantalla.

RESET
Salto incondicional al Monitor. Para volver a Applesoft utilizar CTRL-C o ØG.

CONT
Continúa la ejecución del programa detenido por STOP, END o CTRL-C.

CALL X
Llama a la subrutina en lenguaje máquina que comienza en la posición de memoria X.

END
Detiene la ejecución sin ningún mensaje.

HIMEM
Establece un límite superior en memoria disponible a programas BASIC incluyendo almacenamiento de variables.

LOAD
Carga un programa desde cinta.

LUMEM
Establece un límite inferior a la memoria disponible para los programas BASIC para almacenamiento de variables y elementos similares.

NEW
Borra el programa en curso en memoria.

NOTRACE
Interrumpe el seguimiento de la ejecución del programa que se inició por TRACE.

PEEK(X)
Devuelve el contenido de la posición de memoria X.

POKE X, 25
Cambia el contenido de la posición de memoria X al valor 25.

RUN
Ejecuta el programa comenzando por el número de línea inferior.

RUN *n*
Ejecuta el programa comenzando en la línea número *n*.

SAVE
Conserva y guarda un programa en cinta.

STOP
Detiene la ejecución y presenta la línea de interrupción.

USR(X)
Pasa el valor X a la subrutina en lenguaje máquina.

WAIT
Interrumpe un programa hasta que una posición de memoria particular alcanza un estado especificado.

Edición y de formato

CLEAR
Pone todas las variables a cero.

DEL n_1, n_2
Borra las líneas especificadas del programa.

FRE (∅)
Devuelve la cantidad de memoria disponible en ese momento por el usuario.

FLASH
Activa el modo de producción de destellos de video.

HOME
Borra la pantalla y posiciona el cursor en la esquina superior izquierda.

HTAB X
Posiciona el cursor en la columna especificada en la línea de visualización en curso.

INVERSE
Activa el modo de video inverso.

LIST
Lista el programa completo en memoria.

NORMAL
Desactiva los modos de video Inverso y Destellos FLASH/INVERSE.

POS (∅)
Devuelve la posición horizontal real del cursor (0 a 39).

REM
Permite que se coloquen comentarios en el programa para fines de documentación del programa.

SPC(X)
Solamente existe con la sentencia PRINT, pone X espacios entre el último elemento impreso y el siguiente.

SPEED=X
Cambia la rapidez con que salen los caracteres; desde 0 (la más lenta) a 255 (la más rápida).

TAB(X)
Solamente existe con la sentencia PRINT; mueve el cursor a la posición X (1 a 40).

VTAB(Y)
Mueve el cursor a la línea Y (1 a 24).

Sentencias básicas

DATA
Crea una lista de valores que son asignados por sentencias READ.

DEF FN
Define una función numérica de usuario. DEF FN A(X)=X*3+5.

DIM
Reserva espacio en memoria para una matriz (array) o cadena.

DSP
Visualiza los valores que cambian de la variable especificada a medida que avanza un programa en Integer BASIC.

FOR
Comienza un bucle que repite un juego de instrucciones hasta que una variable incrementada automáticamente alcance un cierto valor.

GET A$
Espera a que el usuario escriba un carácter o caracteres sin necesidad de pulsar la tecla RETURN.

GOSUB *n*
Produce la bifurcación a la subrutina que comienza en la línea *n*.

GOTO *n*
Bifurca a la línea *n*.

IF-THEN
Hace que el programa ejecute condicionalmente la instrucción o instrucciones especificadas.

INPUT
Acepta entrada de caracteres desde el teclado u otro dispositivo de entrada, la evalúa y asigna el valor o valores introducidos a la variable o variables.

IN # *n*
Toma la entrada futura del periférico conectado a la ranura 6 en vez del teclado.

LET
La sentencia asigna un valor o una variable especificada. Su uso no es opcional.

NEXT
Finaliza el bucle comenzando por una sentencia FOR.

ON-GOSUB
Bifurca a la/s subrutina/s especificadas por los números de líneas.

ON-GOTO
Bifurcación condicional a las líneas especificadas en la sentencia.

ONERR GOTO *n*
Bifurca al número de línea especificado *n* cuando se presenta un error posterior en un programa.

POP
Elimina la posición de retorno correspondiente a la última sentencia GOSUB realizada.

PRINT
Da salida de caracteres a la pantalla u otros dispositivos de salida.

READ
Asigna cada elemento de una sentencia DATA a la variable correspondiente de la sentencia READ.

RESTORE
Repone el puntero de datos de la lista de elementos DATA al principio de la lista.

RETURN
Hace que el programa bifurque a la sentencia inmediatamente después de la sentencia GOSUB ejecutada más recientemente.

Funciones (numéricas y de cadena)

ABS(X)
Proporcional el valor absoluto de X.

ASC(X$)
Devuelve el código ASCII correspondiente a un carácter especificado.

ATN(X)
Devuelve el arco tangente del argumento X.

CHR$(N)
Devuelve el carácter correspondiente al código ASCII indicado por N.

COS(X)
Devuelve el coseno del ángulo X expresado en radianes.

EXP(X)
Devuelve la potencia del número *e*.

FN
Llama a una función definida por el usuario anteriormente ejecutada.

INT(X)
Entrega la parte entera de X.

LEFT$(A$, N)
Devuelve los N caracteres más a la izquierda de una cadena.

LEN(A$)
Devuelve la longitud de la cadena A$.

LOG(X)
Devuelve el logaritmo natural de un número.

MID$(A$, M, N)
Devuelve una subcadena A$ de longitud M, comenzando en la posición N.

POS(X)
Proporciona la posición de la columna del cursor.

RIGHT$(A$, N)
Devuelve los N caracteres más a la derecha de la cadena.

RND(X)
Proporciona un número aleatorio.

SGN(X)
Determina si un número es positivo o negativo.

SIN
Proporciona el seno del ángulo X en radianes.

SPC(X)
Desplaza el cursor a la derecha un número determinado de posiciones.

SQR(X)
Proporciona la raíz cuadrada de un número positivo.

STR$(N)
Convierte el valor numérico N en una cadena.

TAB(X)
Desplaza el cursor a la derecha, a la posición de columna especificada.

TAN(X)
Proporciona la tangente de un ángulo en radianes.

USR
Bifurca a una subrutina en lenguaje máquina.

VAL(A$)
Convierte una cadena en una valor numérico.

Funciones numéricas derivadas

Aunque la siguiente lista de funciones derivadas no es, bajo ningún concepto, completa, proporciona alguna de las fórmulas más frecuentemente necesitadas. Ciertos valores de x invalidarán alguna función (por ejemplo, si COS(x)=0, entonces SEC(x) no es real) y, por tanto, su programa deberá comprobarlo.

Ninguna función derivada funcionará en Integer BASIC.

ARCCOS(x)= −ATN(x/SQR(−x*x+1))+1,5707633.
Proporciona la inversa del coseno de x(ABS(x)<1).

ARCCOT(x)= −ATN(x)+1,5707633
Proporciona la inversa de la cotangente de x.

ARCCOSH(x)=LOG(x+SQR(x*x−1))
Proporciona la inversa del coseno hiperbólico de x(x>=1).

ARCCOTH(x)=LOG ((x+1)/(x−1))/2
Proporciona la inversa de la cotangente hiperbólica de x(ABS(x)>1).

ARCCS(x)=ATN (1/SQR (x*x−1))+(SGN(x)−1)*1,5707633
Proporciona la inversa de la cosecante de x(ABS(x)>1).

ARCCSCH(x)=LOG ((SGN(x)*SQR(x*x+1)+1)/x)
Proporciona la inversa de la cosecante hiperbólica de x(x>0).

ARCSEC(x)=ATN(SQR(x*x−1))+(SGN(x)−1)*1,5707633
Proporciona la inversa de la secante de x(ABS(x)>=1).

ARCSECH(x)=LOG((SQR(−x*x+1)+1)/x)
Proporciona la inversa de la secante hiperbólica de x(0<x<=1).

ARCSIN(x)=ATN(x/SQR(−x*x+1))
Proporciona la inversa del seno de x(ABS(x)<1).

ARCSINH(x)=LOG(x+SQR(x*x+1))
Proporciona la inversa del seno hiperbólico de x.

ARCTANH(x)=LOG((1+x)/(1−x))/2
Proporciona la inversa de la tangente hiperbólica de x(ABS(x)<1).

COSH(x)=(EXP(x)+EXP(−x))/2
Proporciona el coseno hiperbólico de x.

COT(x)=1/TAN(x)
Proporciona la cotangente de x(x< >0).

COTH(x)=EXP(−x)/(EXP(x)−EXP(−x))*2+1
Proporciona la cotangente hiperbólica de x(x< >0).

CSC(x)=1/SIN(x)
Proporciona la cosecante de x(x< >0).

CSCH(x)=2/(EXP(x)−EXP(−x))
Proporciona la cosecante hiperbólica de x(x< >0).

LOG_a(x)=LOG(x)/LOG(a)
Proporciona el logaritmo en base a de x(a>0, x>0).

LOG_{10}(x)=LOG(x)/2,30258509
Proporciona el logaritmo común (en base diez) de x(x>0).

MOD_a(x)=INT((x/a−INT((x/a)) * a+.05) * SGN(x/a)
Proporciona x en módulo a: el resto después de la división de x por a(a< >0).

SEC(x)=1/COS(x)
Proporciona la secante de x(x< >π/2).

SECH(x)=2/(EXP(x)+EXP(−x))
Proporciona la secante hiperbólica de x.

SINH(x)=(EXP(x)−EXP(−x))/2
Proporciona el seno hiperbólico de x.

TANH(x)=−EXP(−x)/(EXP(x)+EXP(−x)) * 2+1
Proporciona la tangente hiperbólica de x.

Hewlett Packard
HP-85, 86, 87

Operadores

Aritméticos:

+	Suma
−	Resta
*	Multiplicación
/	División
^	Exponenciación
MOD	Módulo
\ o DIV	División entera

Relación:

=	Igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
< > o #	Distinto

Lógicos

AND OR EXOR NOT

Operador de concatenación de cadenas: &

Prioridad:

( )
Funciones
^
NOT
*, /, MOD, \ o DIV
+, −
Operadores de relación: =, >, <, >=, <=, < > o #
AND
OR, EXOR

Precisión

Precisión	*Exactitud*	*Rango*
REAL	12 dígitos	± 9.99999999999E±499
SHORT	5 dígitos	± 9.9999E±99
INTEGER	5 dígitos	− 99999 a 99999

Variables

Numéricas: A1, B5, C4
De índice: A(4,6), B(15,20), C(40,6)
De cadena: A1$, C$
Subcadenas: A5$[3,20], B$[10], C$[4,4]

Palabras reservadas BASIC

Ordenes

No programables

AUTO
CONT
DELETE
INIT
LOAD
REN
RUN
SCRATCH
STORE (*equivale a* SAVE)
UNSECURE

Programables

CAT
COPY
CTAPE
ERASETAPE
FLIP
LIST
PLIST
PRINT ALL
REWIND
SECURE

Sentencias

ASSIGN #
BEEP
CHAIN
CLEAR
COM
CRT IS
CREATE
DATA
DEFAULT OFF
DEFAULT ON
DEF FN
DEG
DIM
DISP
DISP USING
END
FN END
FOR-NEXT-STEP
GOSUB
GOTO
GRAD
IF-THEN-ELSE
IMAGE
INPUT
INTEGER
KEY LABEL
LET
LET FN
LOAD BIN
NEXT
NORMAL
OFF ERROR
OFF KEY #
OFF TIMER #
ON ERROR GOSUB
ON ERROR GOTO
ON-GOSUB
ON-GOTO

ON-KEY-GOSUB
ON-KEY-GOTO
ON-TIMER-GOSUB
ON-TIMER-GOTO
OPTION BASE
PAUSE
PRINT
PRINT #
PRINT USING
PRINTER IS
PURGE
RANDOMIZE
READ
READ #
REAL
REM
RENAME
RESTORE
RETURN
SHORT
STOP
STORE BIN
TRACE
TRACE ALL
TRACE VAR
SETTIME
WAIT

Funciones

ABS(X)
ACS(X)
ASN(X)
ATN(X)
ATN2(Y,X)
CEIL(X)
CHR$(X)
COS(X)
COT(X)
CSC(X)
DATE
DTR(X)
EPS
ERRL
ERRN
EXP(X)
FLOOR(X)
FP(X)
INF
INT(X)
IP(X)
LEN(S$)
LGT(X)
LOG(X)
MAX(X,Y)
MIN(X,Y)
NUM(S$)
PI
POS(S1$,S2$)
RMD(X,Y)
RND
RTD(X)
SEC(X)
SGN(X)
SIN(X)
SQR(X)
TAB(X)
TAN(X)
TIME
UPC$(S$)
VAL(S$)
VAL$(X)

AMSTRAD
CPC 464/CPC 664/CPC 6128

Variables

Nombre: Longitud variable con el primer carácter una letra.
El nombre no debe ser una palabra clave.

Tipos:

	Caracteres declarativos	*Ejemplos*
Enteros (−32768 a 32767)	%	N% = 434
Reales (6 cifras significativas)	!	M! = 1234
Cadena (255 caracteres máximo)	$	NOMBRE$

Constantes

Hexadecimales: Prefijo & o &H: &HFA
Binarias: Prefijo &X: &X 101

Definición global de variables

DEFINT Definición de variables enteras
DEFREAL Definición de variables reales
DEFSTR Definición de variables de cadena

Expresiones matemáticas

Operadores aritméticos		*Operadores de relación*	*Operadores lógicos*
Exponenciación	^	=	AND
Multiplicación, división	*, /	<	OR
Suma, resta	+, −	>	XOR
Módulo	MOD	< >	
División entera	\	< =	

Listado alfabético de palabras clave BASIC Locomotive
(Amstrad CPC 464, CPC 664, CPC 6128)

ABS
AFTER
AND
ASC
ATN
AUTO
BIN$
BORDER
CAT
CHAIN
CHR$
CINT
CLEAR
CLG
CLOSEIN
CLOSEOUT
CLS
CONT
COPY
COS
CREAL
CURSOR
DATA
DEC$
DEF
DEF FN
DEFINT
DEFREAL
DEFSTR
DEG
DELETE
DERR
DI
DIM
DRAW
DRAWR
EDIT
EI
END
ENT
ENV
EOF
ERASE
ERL
ERR
ERROR
EVERY
EXP
FIX
FOR
FRE
GOSUB
HEX$
HIMEN
IF
INK
INKEY
INKEY$
INP
INPUT
INPUT #9
INSTR
INT
JOY
KEY
KEY DEF
LEFT$
LEN
LET
LINE
LINE INPUT
LINE INPUT #
LIST
LOAD
LOCATE
LOG
LOG10
LOWER$
MASK
MAX
MEMORY
MERGE
MID$
MIN
MOD
MODE
MOVE
MOVER
NEW
NEXT
NOT
ON
ON BREAK STOP
ON ERROR
ON GOSUB
ON GOTO
ON SQ
OPENIN
OPENOUT
OR
ORIGIN
OUT
PAPER
PEEK
PEN
PI
PLOT
PLOTR
POKE
POS
PRINT
PRINT USING
PRINT #9
RAD
RANDOMIZE
READ
RELEASE
REM
REMAIN
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
ROUND
RUN
SAVE
SGN
SIN
SOUND
SPACE$
SPC
SPEED INK
SPEED KEY
SPEED WRITE
SQ
SQR
STOP
STR$
STRING$
SWAP
SYMBOL
SYMBOL AFTER
TAB
TAG
TAGOFF
TAN
TEST
TESTR
TIME
TROFF
TRON
UNT
UPPER$
VAL
VPOS
WAIT
WEND
WHILE
WIDTH
WIDTH
WINDOW
WINDOW SWAP
WRITE
WRITE #
XOR
XPOS
YPOS
ZONE

MSX

Variables

Nombre: Cualquier número de caracteres (el primero ha de ser una letra).

Tipo:

	Carácter declarativo	*Ejemplos*
Enteros (−32768 a +32767)	%	N% = 123 ; A% = 3256
Simple precisión (6 cifras significativas)	!	Número! = 324.25
Doble precisión (14 cifras significativas)	#	Número# = 123456789.12#
Cadenas de caracteres (255 caracteres máximo)	$	Nombre$ = "Luis García"

Constantes

Hexadecimales:	&HFF	Prefijo, &H
Octales:	&0477	Prefijo, &0
Binarias:	&B101	Prefijo, &B

Definición de variables

DEF tipo letras	
DEFINT N	Todas las variables que comienzan por N y no estén declaradas explícitamente por un carácter (!, %, $, #) son variables enteras.
DEFSNG N,M	Variables de simple precisión, si comienzan por N y M
DEFSTR E,R,T	Declara como variables de cadena todas las variables que comienzan por E, R y T.

Expresiones y operadores

Operadores aritméticos:

^	Exponenciación
*, /	Multiplicación, División
+, −	Suma, Resta
MOD	Módulo
\	División entera

Operadores de relación:

=	Igual
<	Menor que
>	Mayor que
< >	Distinto
< =	Menor o igual
> =	Mayor o igual

Operadores lógicos:

NOT, AND, OR, XOR, IMP, EQV

Prioridad de operadores:

1. Paréntesis
2. Operadores aritméticos
3. Operadores relacionales
4. Operadores lógicos

Listado de instrucciones BASIC

Ordenes del sistema

AUTO, CONT, DELETE, LIST, LLIST, NEW, RENUM, RUN, TRON-TROFF

Sentencias generales

CLEAR
DATA
END
INPUT
LET
LINE INPUT
LPOS(O)
LPRINT
LPRINT USING
OUT
POKE
PRINT
PRINT USING
READ
REM
RESTORE
STOP
SWAP
WAIT

Declaraciones de variables

DEF FN
DEFDBL
DEFINT
DEFSTR
DEFUSR
DIM
ERASE

Bucles y bifurcaciones

FOR-NEXT-STEP
GOSUB
GOTO
IF-THEN-ELSE
INTERVAL ON/OFF/STOP
ON-GOSUB
ON-GOTO
ON STOP GOSUB
RETURN
STOP ON/OFF/STOP

Pantalla

CLS
COLOR
LOCATE
SCREEN
VDP
VPEEK
VPOKE
CSRLIN
POS

Tratamiento de errores

ON ERROR GOTO
ERROR

ERL
ERR
RESUME

Teclado

KEY
KEY LIST
KEY ON/OFF
ON KEY GOSUB
KEY(N) ON/OFF/STOP

Palancas de juego

ON STRIG GOSUB
STRIG(N) ON/OFF/STOP
STRIG(N)
STICK(N)
PDL(N)
PAD(N)

Gráficos

CIRCLE
DRAW
LINE
PAINT
PRESET
PSET

«Sprites»

SPRITE$( )
SPRITE ON
SPRITE OFF
SPRITE STOP
ON SPRITE GOSUB
PUT SPRITE

Ordenes de tratamiento de casetes

BSAVE
BLOAD
CSAVE
CLOAD
CLOAD?
LOAD
MERGE
SAVE
MOTOR
MOTOR ON
MOTOR OFF

Ficheros secuenciales

OPEN
PRINT#
INPUT#
LINE INPUT#
EOF
CLOSE
MAXFILES

Sonidos

BEEP
PLAY
PLAY(U)
SOUND

Funciones

Aplicación general

BASE
BIN$
CDBL
CINT
CSNG
CVD
CVI
CVS
DSKF
FIX
FN
FRE
HEX$
INKEY$
INP
INPUT$
INT
LOF
LPOS
OCT$
PEEK
POINT
RND
SPC
TAB
TIME
USR
VARPTR
WIDTH

Matemáticas

ABS
ATN
COS
EXP
LOG
SGN
SIN
SQR
TAN

De cadena

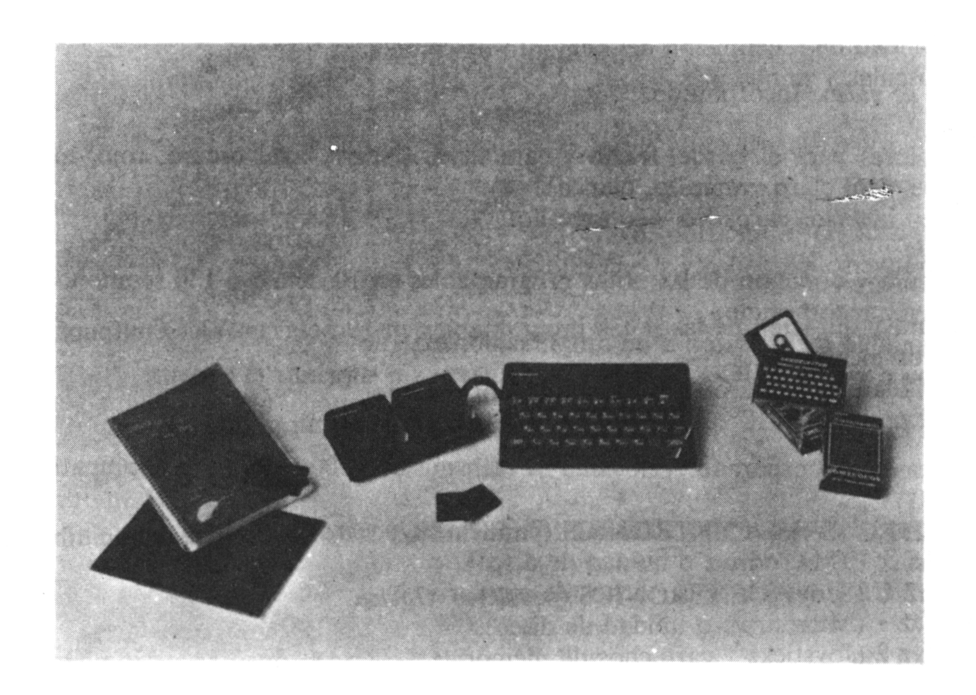
ASC
CHR$
INSTR$
LEFT$
LEN
MID$
MKD$
MKI$
MKS$
RIGHT$
SPACE$
STR$
STRING$
VAL

Formatos de las instrucciones de MSX-BASIC

AUTO comienzo, incremento
BEEP
BLOAD nomfichero, opcejec, offset
BSAVE nomfichero, dircom, dirfin, direjec
CALL (o _) nomprograma
CALL FORMAT
CALL SYSTEM
CIRCLE STREP (x, y), radio, color, angcom, angfin, aspecto
CLEAR tamcad, esplibre
CLOAD nomfichero
CLOSE nomfichero
CLS
COLOR primerplano, fondo, contorno
CONT
COPY nomfichero TO nomfichero 2
CSAVE nomfichero, velocidad
DATA valores
DEF FN var (args) = expresión
DEF tipoval rangos
DEF USR digit = n
DELETE rango
DIM vars
DRAW cadgml
END
ERASE arrays
ERROR n
FIELD # n, anchural AS varl$, ...
FILES nomfichero
FOR var = valcom TO valfin STEP incremento
GET # n, numrec
GOSUB línea
GOTO línea
IF condición THEN sentencia
IF condición THEN sentencia1 ELSE sentencia2
INPUT prompt, vars
INPUT # n, vars
INTERVAL OFF/ON/STOP
KEY LIST
KEY n, cadena
KEY (n) OFF/ON/STOP
KILL nomfichero
LET var = expresión
LFILES nomfichero
LINE INPUT prompt; var$
LINE INPUT # n, var$
LINE STEP (x1, y1) — STEP (x2, y2), color, B
LINE STEP (x1, y1) — STEP (x2, y2), color, BF
LIST rango
LLIST rango
LOAD nomfichero, R
LOCATE x, y, cursor
LPRINT vars
MAXFILES = n
MERGE nomfichero

MOTOR OFF
MOTOR ON
NAME nomfichero AS nomfichero2
NEW
NEXT var
ON ERROR GOTO línea
ON INTERVAL = n GOSUB línea
ON KEY GOSUB líneas
ON línea GOSUB líneas
ON línea GOTO líneas
ON SPRITE GOSUB línea
ON STOP GOSUB línea
ON STRIG GOSUB líneas
OPEN dispositivo FOR modo AS # n
OUT dirección, valor
PAINT STEP (x, y,), color, contorno
PLAY cadensmml
POKE dirección, valor
PRESET STEP (x, y,), color, contorno
PRINT # n, USING cadformato, vars
PSET STEP (x, y,), color
PUT # n, numrec
PUT SPRITE n, STEP (x, y), color, imagen
READ vars
REM texto
RENUM nuevalínea, antigualínea, incremento
RESTORE línea
RESUME modo
RETURN
RUN nomfichero
SAVE nomfichero, A
SCREEN modo, tamsprite, pulstecla, velocinta, tipoimpr
SOUND registro, valor
SPRITE OFF/ON/STOP
STOP
STOP OFF/ON/STOP
STRIG OFF/ON/STROP
SWAP var1, var2
TROFF
TRON
VPOKE dirección, valor
WAIT dirección, andexp, xorexp
WIDTH n

ZX Spectrum/+/128K



CARACTERISTICAS TECNICAS

Unidad central

Microprocesador: Z80A de Zilog de frecuencia 3.5 MHz.

Memoria ROM: 16K conteniendo BASIC, sistema operativo y generador de caracteres.

Memoria RAM: 16K (8×4116). Ampliación a 48K en opción.

Teclado

Tipo: Mecánico, QWERTY.

Teclas: 40 autorrepetitivas y testigo sonoro. Mayúsculas y minúsculas. Palabras reservadas BASIC por simple pulsación. Cada tecla representa 4, 5 ó 6 funciones diferentes. Tres modos de trabajo: h (escritura en minúsculas), K (palabras reservadas), C (escritura en mayúsculas), G (caracteres gráficos), E (acceso a órdenes y funciones en las teclas).

Presentación: 24 líneas de 32 caracteres (24 × 32 = 768 caracteres).

22 líneas para visualización normal de textos (datos, resultados o programas).
2 últimas líneas para entrada, edición y visualización de mensajes de error.
1 carácter = 8 × 8 pixels (puntos).

Gráficos de alta resolución: 256 × 192 = 49152 pixels (puntos) direccionables individualmente.

Sentencias y órdenes BASIC de dibujo: punto, vector, círculo, arco (se pueden mezclar textos y gráficos).

Generador de caracteres: Código ASCII ampliado (matriz 8 × 8).

21 caracteres programables (definidos por èl usuario).
16 caracteres gráficos incorporados.
Posibilidad de redefinición del conjunto de caracteres.
Mayúsculas y minúsculas.

Modulador de vídeo: Incorporado.

Color: 8 colores para el borde, fondo y caracteres. (Negro, azul oscuro, rojo, magenta, verde, azul claro, amarillo, blanco).
Vídeo inverso, parpadeo y sobrebrillo.

Sonido: Altura y duración de las notas programables en 10 octavas: 130 semitonos.
Altavoz incorporado.
Posibilidad de conexión a un amplificador externo.
Toma o clavija MIC puede recibir un casco sin suprimir el altavoz.

Interfaces:

De casete incorporado. Velocidad de transmisión 1500 baudios (no compatible con ZX81).
RS 232 C (Serie). CENTRONICS (paralelo). *Opción.*
Interfaz 1 (Microdrive o unidad de disco).
Interfaz 2 (Joysticks y cartuchos de memoria).

Memoria externa:

Casete: Cinta de casete. Dispone de entrada/salida para la conexión directa a un magnetófono a casete comercial.
Discos-flexibles: Microdisquetes de 2,5" y 85K. Velocidad de transferencia 160K/segundo. Se pueden conectar 8 unidades de microdisquetes.
Sistema operativo:
Sinclair.

Periféricos:

Impresora ZX (compatible ZX81).
Unidades de cartuchos (Microdrives).
Periféricos serie y paralelo mediante interfaz serie/paralelo.

Operaciones aritméticas y lógicas

Símbolo	*Significado*	*Prioridad*
Funciones	Asignación de subíndices y fragmentación salvo NOT y operación unaria	12
↑	Potenciación	11
–	Operación unaria «menos» (usada para negar algo)	10
*, /	Mutiplicación, división	9
+, –	Suma, resta	8
=	Igual a	6
>=	Mayor que	5
<=	Menor que	5
=	Menor o igual que	5
>=	Mayor o igual que	5
<>	Distinto	5
NOT	«NO» inversión lógico	4
AND	«Y» lógico	3
OR	«O» lógico	2

Significado letras minúsculas en sentencias

Sentencias y órdenes

BEEPx,y
Sonido de duración, x segundos, y semitonos (sensible a BREAK).
BEEP x,y

BORDER
Color del contorno.
BORDER 0.

BRIGHTb
Orden de brillo de la pantalla. Efecto permanente solo
b=0 normal
b=1 brillo especial
b=8 transparente
BRIGHT 1

CAT
Catálogo de los ficheros de la unidad de disco (microdrive).

CIRCLEx,y,z
Círculo de centro (a,b) y radio C.
CIRCLE 4.5, 6,15.

CLEAR
Anulación de variables.

CLEARn
Anulación de variables, pero de ser posible cambia la variable RAMTOP a la dirección
23821 n 3276 versión base 16K
23821 n 65535 versión de 48K.

CLOSE#
Cierre de fichero (no funciona sin microdrive).

CLS
Borrado o limpiado de pantalla.

CONTINUE
Continúa la ejecución del programa en la línea donde se interrumpió o en la siguiente.

COPY
Reproduce el contenido de las 22 primeras líneas de la pantalla en la impresora si está conectada; en caso contrario no hace nada.

DATA a, b, c
Declaración de una lista de datos. Debe entrar incluida en el programa.

DEF FN a(a, ..., v)=e
Definición de función numérica definida por el usuario, donde e es una expresión.
Nombre de la función: una de las 26 letras del alfabeto.
Como máximo 52 argumentos (letra o letras seguidos de $).

DEF FN a$ (a, ... v)=e
Igual pero para una función de cadena.

DELETE f
Borrado de un fichero sólo funciona con microdrive (microunidad de disquete).

DIM a(n_1, n_2 ... n_k)
Definición y dimensionado de una tabla de K dimensiones e inicialización a cero. Los índices comienzan en el valor 1.
DIMa (4,5) definición de una tabla de dimensiones 4×5.

DIM a$ (n_1, n_2 ... n_k)
Igual pero para una tabla alfanumérica o de cadenas.

DRAW x,y
Trazado de una resta a partir de la posición actual del cursor hasta el punto de coordenadas X, Y.
PLOT 0,50: DRAGON 80, −50.

DRAW x,y,z
Trazado de un arco a partir de la posición actual del cursor hasta el punto de coordenadas X, Y mientras gira a través de un ángulo Z.

ERASE
Borrado de un microdrive.

FLASH
Parpadeo de caracteres
n=0 estable; n=1 parpadeo; n=8 falta de cambio
FLASH n

FOR a=x TO y
Equivale a FOR a=X TO STEP 1.

FOR a=x TO y STEP z
Definición de un bucle, inicio en el valor x; límite en y e incremento o paso z.
FOR a=1 TO STEP 2; FOR a=5 TO 1 STEP −1

FORMAT
Formateado o inicialización de un microdrive (no opera sin microdrive).

GOSUB n
Llamada a la subrutina que comienza en la línea n, retorno a la siguiente línea a GOSUB después de encontrar RETURN GOSUB-500.

GOTO n
Salto incondicional a la línea n (si no existe a la primera línea después de ella).
GOTO 300.

IF × THEN s
Si × es verdadero (no cero) entonces se ejecuta 5.
IF × THEN GOTO 10; IF a=3 THEN PRINT y

INK n
Establece el color del primer plano de los caracteres n=0 a 7; uno de los ocho colores del Spectrum.
n=8 color transparente
n=9 constrante . color blanco
Si el fondo es azul, negro, rojo o magenta.
Color negro si el fondo es verde, ciano, amarillo, blanco.
INK 7

INPUT
Parada voluntaria en el programa para introducir datos.
INPUT «QUÉ EDAD TIENES», edad.

INVERSE n
Orden de inversión de caracteres.
n=0 impresión en video normal: color del papel sobre color de la tinta.
n=1 caracteres se imprimen en video inverso.
INVERSE 1

INVERSE n
Orden de video inverso. El efecto es permanente si se utiliza solo.
INVERSE 0: Video normal. Color de la tinta sobre el color del papel.
INVERSE 1: Video inverso. Color del papel sobre el color de la tinta.

LET
Asignación de un valor a una variable.
LET b=5
LET b$=«computadora»

LISTn
Listado del programa BASIC comenzando en la línea n;
LIST 40

LIST
Equivalente a LIST ∅

LLIST n
Igual que LIST pero imprimiendo en impresora.

LIST
Equivalente a LLIST.

LOAD f
Carga de un programa y sus variables
LOAD"" LOAD"INVASORES"

LOAD f DATA ( )
Carga de una matriz o tabla numérica.
LOAD «test» DATA A ( )

LOAD f DATA$ ( )
Carga una matriz de cadenas
LOAD «test» DATA b$ ( )

LOAI f CODE m,n
Carga, como máximo, n octetos a partir de la dirección m.
Igual que la anterior, pero sólo se indica la dirección inicial.
LOAD «octetos» CODE 32500

LOADf SCREEN$
Carga la pantalla.
LOAD «pantalla» SCREEN$

LPRINT
Como PRINT, pero utilizando la impresora.

MERGE
Como LOADf, pero sin borrar el programa antiguo (salvo las nuevas líneas o variables con el mismo nombre o número).

NEW
Borra el programa antiguo BASIC. Reinicializa el sistema. No afecta a las variables del sistema: RAMTOP, P-RAMP, RASP, PIP, UDG ni los caracteres gráficos del usuario.

NTEv
Final de un bucle. Se produce el error 2 si no hay ninguna variable.

OPEN#
Apertura de un fichero. Sólo funciona con Microdrive.
10 OPEN# 1, «M»
20 PRINT # 1, «TEST»

OUTm,n
Salida del octeto n en el port m (carga el par de registros bc con m y el registro a con n y realiza la instrucción de lenguaje ensamblador: out(c),a.
OUT 254,7 activación del altavoz
OUT 254,23 desactivación del altavoz
Los ports están numerados del 0 al 255.
El port 254 controla el color del borde y la salida del sonido.

OVER n
Sobreimpresión de caracteres
n=0 borrado de los antiguos caracteres.
n=1 los nuevos caracteres se mezclan con los antiguos.
100 DRAW OBER 1

PAPER n
Igual que INK, pero controlando el color de fondo.
PAPER 4

PAUSE
Interrumpe la ejecución del programa y produce una pausa durante n/50 segundos (50, frecuencia de la red eléctrica en Europa, 60 en USA, n/60) o hasta que se pulse la tecla. n=50, retardo 15. 0 n 65535, si está fuera de estos límites se producirá error .n=0 la pausa dura hasta que se pulsa una tecla.
PAUSE 200.

PLOT c,x,q
Dibuja un punto (pixel) de coordenadas x,y en la pantalla; c es opcional e indica el color.

POKE m,n
Escribe el octeto n en la dirección de memoria m 0 m 65535 −255 n 255. Fuera de estos límites se produce el error B.
No se puede escribir en los primeros 16K de ROM POKE 24675, 200.

PRINT
Control de la salida; después de PRINT se puede indicar los siguientes elementos:
variables
constantes
cadenas (entre comillas)
expresiones (numéricas)

Los separadores que se pueden utilizar
; ningún espacio
, se utiliza como tabulación
Ordenes de posición
ATm,n visualización en la instrucción de la línea m y la columna n.
TAB n tabulación de control.
Ordenes de presentación
PAPER, FLASH, BRIGHT, INVERSE, OVER

RANDOMIZE
Controla el generador de números aleatorios o la semilla; números aleatorios diferentes se generan cada vez.

RANDOMIZE n
Si n=0 equivalente a RANDOMIZE 0. Si n=entero se generará el mismo número aleatorio a partir de n.

READ v_1, v_2 ... v_k
Asignación a las variables v_1, v_2, ..., v_{ic} de los valores declarados en las sucesivas sentencias DATA.

REM
Sin efecto. Sentencia para insertar comentarios.
10 REM comentarios.

RESTORE n
Restaura el puntero de DATA a la primera sentencia DATA de la línea n. donde se efectuará la primera lectura de la próxima sentencia READ.

RESTORE
Equivalente a RESTORE 0.

RETURN
Fin de una subrutina a la que se bifurcó con la sentencia GOSUB.
100 RETURN.

RUN n
Equivalente a RUN 0.

RUN n
Borrado (CLEAR) y después GOTOn.

SAVE f
Conserva (guarda) el programa y variables en casete.
SAVE test.

SAVE f LINE m
El programa comenzará automáticamente en la línea n si el programa está cargado.
SAVEf, después de la carga.

RUN n, automáticamente.
SAVE text LINE 10.

SAVE f LINE
Igual que la anterior pero RUN 0 automáticamente.

SAVE f DATA ( )
Conserva una matriz numérica.

SAVE f DATA$ ( )
Conserva matriz de cadenas.

SAVE f CODEm,n
Conserva n octetos a partir de la dirección m.

SAVE f SCREEN$
Conserva la pantalla.

STOP
Detención de la ejecución del programa. CONTINUE reanudará la ejecución con la siguiente sentencia.

VERIFY
Verifica si el programa «f» se ha cargado correctamente en la cinta de casete.

Funciones

ABS (X)
Calcular valor absoluto de X.

ACS (X)
Calcula el arco seno de X (−1 X 1)

AND
Operación lógica AND («Y») .ANDb
a ANDb=a si b<>0
o si b=0
a$ AND b=0 si b=0
a$ si b<>0.

ASN (X)
Calcula arco seno en radianes.

ATN (X)
Calcula arco tangente en radianes.

ATTRx,y
Determina los atributos de un pixel (punto) semigráfico en la pantalla. X línea y columna.
$0 \leqslant x \leqslant 23$ $0 \leqslant y \leqslant 31$
Z=ATTR (4,6)

bit 7	bit 6
1=parpadeo	1=sobrebrillo
0=fijo	0=normal

bits 5, 4, 3		bits 2, 1, 0
000	Negro	000
001	Azul	001
010	Rojo	010
011	Magenta	011
100	Verde	100
101	Ciano	101
110	Amarillo	110
111	Blanco	111
PAPEL		INK

BIN
Convierte un número digital en binario a entero.
a=BIN 101 a=5

CHR$ (X)
Obtiene el carácter correspondiente al código X (o el entero redondeado más próximo).

CODE (cadena)
Entrega el código ASCII decimal del primer carácter de la cadena (0 si la cadena es vacía).

COS (X)
Calcula coseno de x radianes.
A=cos (PI/6)

EXP (X)
Calcula e^x (exponencial de x).

FN
Indica una función predefinida; FN está seguida por una letra. Permitidas 26 funciones numéricas o de cadenas previamente definidas por DEF.
PRINT FN p(2,3), FNq(a)

IN (X)
Lectura del puerto n. Carga el par de registros bc con x y la instrucción en lenguaje ensamblador en a (c).

INKEY$
Explora el teclado, detectando si se ha pulsado una tecla y asignándola a INKEY$
b=INKEY$

INT (X)
Parte entera de x (se redondea por defecto).
PRINT INT (3.46) 3
PRINT INT (−4.25) −5

LEN (cadena)
Determina la longitud (número de caracteres) de una cadena.
b=LEN (a$)

LN (X)
Calcula logaritmo neperiano de x (x>0)
b=LN (6*x)

NOTx
Negación lógica.
NOT x=0 (falsa) si x es verdadera (no cero)
NOT x=1 (verdadera), si x es falsa (cero)

OR
Operación lógica OR («o»)
a OR b=1 (verdadera) si b es verdadera (no cero)
=x, si y es falsa (cero)

PEEKa
Determina el valor entero de la dirección decimal a (0 a 65535). Error B si está fuera del margen.
PEEK 1044.

PI
Valor de π, 3.14159265.

POINT (x,y)
Indica si el pixel (punto) es color de la tinta (1) o color del papel (0).
$0 \leqslant x \leqslant 255$ $0 \leqslant y \leqslant 175$

RND
Genera un valor aleatorio.

SCREEN$ (x,y)
Determina la posición del carácter en la pantalla (fila x, columna y).
$0 \leqslant x \leqslant 23$ $0 \leqslant y \leqslant 31$

SGN (x)
Calcula el signo x
x<0 SGN (x)=−1
x>0 SGN (x)=+1 *si* x=0 SGN(x)=0
SGN (4*A/3)

SIN (x)
Calcula seno de x en radianes.

SQR (x)
Calcula la raíz cuadrada de x.
x>0
a=SQR (b 2−4a*c)

STR$ (x)
Convierte el número en una cadena.
PRINT STR$ 464

TAN (x)
Calcula la tangente del ángulo x en radianes.
a=TAN (4.65*b)

USR (x)
Llamada a una subrutina en código máquina cuya dirección inicial es x. En el retorno de la ejecución, el resultado es el contenido del par de registros bc.

USR$ (cadena)
Devuelve la dirección de uno de los 21 caracteres gráficos del usuario. Ocho octetos consecutivos definen un carácter.

VAL (cadena)
Evalúa la cadena de caracteres como una expresión numérica.
VAL «2*3»=6

VAL$ (cadena)
Evalúa una cadena de caracteres como una cadena.
VAL+« « « chocolate « « «=«chocolate»

Colores

0 Negro
1 Azul
2 Rojo
4 Magenta
5 Verde
6 Ciano
7 Amarillo
8 Blanco

Mensajes de error

Código	*Mensajes*	*Error*
0	OK	OK-Correcto
1	NEXT whithout FOR	NEXT sin FOR
2	Variable not found	Variable no hallada (no definida)
3	Subcript wrong	Subíndice o subscrito erróneo
4	Out of Memory	Desbordamiento de la capacidad de memoria
5	Out of screen	Fuera de pantalla (escritura más allá de la línea 22.ª de la pantalla)
6	Number too big	Número demasiado grande (10^{38})
7	RETURN whithout GOSUB	RETURN sin GOSUB
8	End of file	Final de fichero
9	STOP	Sentencia STOP
A	Invalidid argument	Argumento erróneo
B	Integer out of range	Entero fuera del rango permitido
C	Nonsense in BASIC	Sin significado en BASIC
D	BREAK-CONT repeats	Se repite BREAK durante alguna operación con los periféricos
E	Out of DATA	Faltan datos.
F	Invalid file name	Nombre de fichero incorrecto
G	No room for line	Desbordamiento de la memoria ocasionado por la última línea introducida
H	STOP in INPUT	STOP en los datos de entrada
I	FOR without NEXT	FOR sin NEXT
J	Invalid I/O Device	Dispositivo de entrada/salida no válido (error en dispositivo)
K	Invalid colour	Color incorrecto
L	BREAK into Program	Existe un BREAK en el programa
M	RAMTOP no good	RAMTOP incorrecto
N	Statement lost	Sentencia pérdida
O	Invalid Stream	Serie de caracteres incorrectos
P	FN without DEF	FN sin DEF
Q	Parameter error	Error de parámetro
R	Tape loading error	Error de carga en la cinta

Commodore 64/Commodore 128

Sentencias BASIC

CLOSE
Cierra un fichero lógico.
CLOSE 14 *Cierra el fichero lógico 14.*

CLR
Pone todas las variables numéricas a cero y asigna valores nulos a todas las variables de cadena. Se libera todo el espacio de memoria de las matrices.

CMD
Envía todas las salidas que habrían ido a la pantalla a otra unidad específica.
CMD 5 *Dirige la salida a la impresora.*

CONT
Restablece la ejecución de un programa después de una ruptura (BREAK).

DATA
Define constantes que son asignadas a variables mediante sentencias READ.

DEF FN
Permite definir funciones de propósito especial y utilizarlas en programas.

DIM
Reserva espacio de memoria para las variables de una matriz.

END
Termina la ejecución de un programa.

FOR-NEXT-STEP
Todas las sentencias comprendidas entre la sentencia FOR y la sentencia NEXT se vuelven a ejecutar un número determinado de veces.

GET
Recibe caracteres por separado como entrada desde el teclado.
GET *variable:* GET C; GET C$.

GET #
Recibe caracteres por separado como entrada desde un dispositivo de almacenamiento externo identificado mediante el número de fichero lógico.

GOSUB
Bifurca la ejecución de un programa a una línea especificada *(subrutina)*.

GOTO
Bifurca incondicionalmente a una línea especificada.

IF-THEN
Proporciona la ejecución condicional de sentencias basándose en una expresión relacional.

INITIALICE
Inicializa un disquete.

INPUT
Recibe datos de entrada desde el teclado.

INPUT#
Introduce uno o más datos desde un dispositivo externo identificado mediante el número de fichero lógico.

LET
Asigna un valor a una variable determinada.

LIST
Visualiza una o más líneas de un programa.

LOAD
La sentencia LOAD carga un programa desde un dispositivo externo a memoria.

NEW
Borra el programa actual en la memoria.

ON-GOSUB
Suministra una llamada condicional a una subrutina, de entre varias dentro de un programa, dependiendo del valor actual de una variable.

ON-GOTO
Produce un salto condicional a uno de varios puntos de un programa, dependiendo del valor actual de una variable.

OPEN
Abre un fichero lógico y prepara el dispositivo físico asignado.

POKE
Almacena un octeto de dato en una posición de memoria determinada.

PRINT
Visualiza datos en la pantalla.

PRINT#
Imprime en un dispositivo externo (cinta, disco o impresora).

READ
Asigna valores de una sentencia DATA a las variables nombradas en la lista de parámetros de READ.

REM
Permite colocar comentarios en el programa con propósitos de documentación.

RESTORE
Restablece el puntero de la sentencia DATA al comienzo de los datos.

RETURN
Bifurca el control del programa a la siguiente sentencia de la llamada GOSUB más reciente. Toda subrutina tiene que terminar con una sentencia RETURN.

RUN
Comienza la ejecución del programa actualmente almacenado en memoria. Cierra todos los ficheros abiertos e inicializa todas las variables a 0 o valores nulos.

SAVE
Escribe una copia del programa contenido en la memoria en un dispositivo externo.

STOP
Produce la detención de la ejecución del programa.

VALIDATE
Se valida el disquete de una unidad determinada.

VERIFY
Compara el programa actual de memoria con el contenido de un fichero de programas.

WAIT
Detiene la ejecución del programa hasta que una posición de memoria especificada adquiera un valor especificado.

Funciones BASIC

ABS
Proporciona el valor absoluto del argumento.

ASC
Proporciona el número del código ASCII del carácter especificado.

ATN
Proporciona el arco tangente del argumento (en radianes).

CHR$
Proporciona la representación de cadena del código ASCII especificado.

COS
Proporciona el coseno del argumento.

EXP
Proporciona el valor de $e^{argumento}$. EXP *(argumento)*.

FRE
Entrega los octetos libres en la memoria y disponibles para el usuario
FRE(1) *Imprime el número de octetos libres.*

INT
Proporciona la parte entera del argumento.

LEFT$
Proporciona el carácter o caracteres más a la izquierda de una cadena.

LEN
Proporciona la longitud de la cadena argumento.

LOG
Proporciona el logaritmo natural del argumento (logaritmo en base *e*).

MID$
Proporciona cualquier parte especificada de una cadena.

PEEK
Proporciona el contenido de una posición de memoria especificada.

POS
Proporciona la posición de la columna del cursor.

RIGHT$
Proporciona el/los carácter/es más a la derecha de una cadena.

RND
Genera secuencias de números aleatorios comprendidos entre 0 y 1.

SGN
Determina si un número es positivo, negativo o cero.

SIN
Devuelve el seno del argumento.

SPC
Desplaza el cursor hacia la derecha en el número de posiciones especificado en el argumento.

SQR
Proporciona la raíz cuadrada de un número positivo. Un número negativo producirá un mensaje de error.

ST
Proporciona el valor actual del estado de la E/S.

STR$
Proporciona la cadena equivalente de un argumento numérico.

SYS
Es una función del sistema que transfiere el control del programa a un subsistema independiente.

TAB
Desplaza el cursor a la derecha hasta la posición de la columna especificada.

TAN
Proporciona la tangente del argumento.

TI, TI$
Representan dos variables que representan la hora real en el reloj de la computadora.

USR
Es una función del sistema que pasa un parámetro a una subrutina escrita por el usuario en lenguaje ensamblador cuya dirección está contenida en las posiciones 1 y 2 de la memoria. USR también produce un parámetro de retorno desde la subrutina.

VAL
Proporciona el equivalente numérico de la cadena argumento.

Dragón 32/64/200 (MBASIC)

Operadores lógicos y matemáticos

Símbolo	*Operación*	*Prioridad*
^	Potenciación	1
–	Menos unario	2
*	Multiplicación	3
/	División	3
+	Suma	4
>	Mayor que	5
<	Menor que	5

Símbolo	*Operación*	*Prioridad*
=	Igual a	5
< >	Distinto de	5
> =	Mayor o igual que	5
< =	Menor o igual que	5
NOT	Negación lógica	6
AND	Y lógico	6
OR	O lógico	6

El único operador matemático aplicable con cadenas es + e indica la concatenación.
Los operadores de relación pueden utilizarse para comparar cadenas al igual que elementos numéricos.

Teclas de control

[< –]
Retroceso. Elimina el último carácter.

[SHIFT][+]
Borra la línea actual.

[BREAK]
Interrumpe el programa en ejecución y devuelve control al teclado.

[CLEAR]
Limpia la pantalla.

[ENTER]
Retorno de carro, fin de la línea actual.

[SHIFT][e]
Detiene temporalmente la ejecución de un programa; continuará al pulsar una tecla.

[SHIFT][0]
Conmuta entre mayúsculas y minúsculas.

[ESPACIO]
Introduce un carácter en blanco.

Sentencias BASIC

CLEAR n,h
Reserva n bytes de espacio para cadenas y borra todas las variables; h especifica la dirección BASIC superior.
CLEAR 500.

CLS c
Limpia la pantalla con el color especificado.
0. Negro.
1. Verde.
2. Amarillo.
3. Azul.
4. Rojo.
5. Beige.
6. Turquesa.
7. Magenta.
8. Naranja.

DATA
Almacena datos en un programa para usarlos en sentencias READ DATA 4, 7, 14.2, SUJETO.

DEF FN
Define una función numérica de usuario.
DEF FNA(X)=X*X+3*X.

DEFUSR n
Define un punto de entrada para la función n de usuario, n=0−9 DEFUSR2=14000.

DIM
Dimensiona una o más matrices, DIM X(40), A$(7,6), B(10,2).

END
Termina la ejecución de un programa.
END.

EXEC dir
Transfiere el control a un programa en lenguaje máquina que empieza en dir. Si se omite la dirección, va al último ajuste de CLOADM.
EXEC 45043.

FOR TO NEXT STEP
Crea un bucle en el programa que se ejecuta para el rango de valores especificado; STEP indica el incremento. Si se omite STEP se toma 1.
FOR X=1 TO 10...NEXT X.
FOR A=−1.3 TO 7.6 STEP 0.1.
FOR G=50 TO 10 STEP −10...NEXTG.

GOSUB
Llama a la subrutina que empieza en la línea indicada.
GOSUB 500.

GOTO
Provoca el salto inmediato del programa al número de línea especificado.
GOTO 45.

IF condición THEN acción1 ELSE acción2
Si la condición es cierta se realiza la acción1, si es falsa la acción2.
IF A<3 THEN 200.
IF B−C>0 THEN X=X+1 ELSE Y=Y−1.

INPUT
Detiene la ejecución y espera datos por el teclado.
INPUT A, B, C, D.

LET
Asigna valores a variables (opcional).
LET X=47.4.

LINE INPUT
Permite la entrada de una línea por el teclado, incluyendo comas. La línea se termina con [ENTER].
LINE INPUT "TITULO"; T$.

ON..GOSUB
Bifurca a una de las posibles líneas de comienzo de subrutina.
ON I GOSUB 100, 200, 300.

ON..GOTO
Salto condicional a una de las líneas indicadas.
ON K GOTO 245, 187, 310.

POKE dir, valor
Pone el valor en la dirección de memoria. El valor debe estar entre 0-255.
POKE 27852,0.

PRINT
Presenta el contenido de la lista por pantalla. Una (,) continúa la impresión a 16 columnas. Un (;) mantiene la posición de impresión.
PRINT "LA RESPUESTA".
PRINT A, B.
PRINT "HA TENIDO"; T: "INTENTOS".

PRINT TAB
Mueve el cursor a la columna especificada.
PRINT TAB (10); "INTERES".

PRINT USING
Presenta la salida con el formato especificado:
campo numérico$ signo $ delante de números.
** rellena los espacios al principio con asteriscos.
^^^^ Formato exponencial
%espacios% Campo de cadena
longitud de espacios+2.
+ Hace que se imprima el signo.
PRINT USING «##.##»; A, B.
PRINT USING «% %»; A$.

PRINT e posición
Imprime en la posición especificada de la pantalla (0-511).
PRINT e8, "PAGINA", N.

READ
Asigna el siguiente elemento de una sentencia DATA a la variable correspondiente.
READ A, B, C$.

REM
Permite introducir comentarios en un programa, se ignora todo lo que venga detrás.
REM ESTO ES UN COMENTARIO.

RESTORE
Posiciona el puntero de datos en el primer elemento de la primera sentencia DATA.
RESTORE.

RETURN
Provoca el retorno de una subrutina a la sentencia que sigue a GOSUB.
RETURN.

STOP
Detiene la ejecución del programa en esta línea.
Utilice CONT para seguir.
STOP.

Sentencias de generación de sonidos

PLAY *música*
Toca la cadena musical formada por lo siguiente:
A-G (o 1-12), notas
On, octava n=1 a 5
Vn, volumen n=1 a 31
Ln, duración de nota n=1 a 255
Tn, tempo n=1 a 255
Pn, pausa n=1 a 255
XA$; ejecuta la subcadena A$.
Sostenidos (#) o bemoles (−).
PLAY"03L2GBØ4CXY$;"

SOUND *tono, duración*
Suena el tono especificado (1 a 255) durante el tiempo dado.
SOUND 180,5.

AUDIO
Conecta o desconecta la salida al altavoz del televisor.
AUDIO ON.
AUDIO OFF.

CLOAD
Carga un programa desde el cassette. Se encargará el primer fichero encontrado, a menos que se especifique un nombre.
CLOAD.
CLOAD "NOMBRE".

CLOADM
Carga un programa en lenguaje máquina del cassette. Se puede indicar un desplazamiento sobre la dirección de carga.
CLOADM.
CLOADM "NOMBRE".
CLOADM "NOMBRE", 2500.

CLOSE
Cierra los ficheros abiertos.
CLOSE.
CLOSE #−1.

CSAVE
Graba un programa en cassette (el nombre debe tener un máximo de 8 caracteres). Si se pone una A, el programa se grabará en formato ASCII.
CSAVE "NOMBRE".
CSAVE "NOMBRE", A.

CSAVEM *nombre, comienzo, fin, longitud*
Graba en el cassette un programa en lenguaje máquina.
CSAVEM "NOMBRE", 4E, 6F, 5F.

EOF(−1)
Es cierto si se ha leído el final del fichero.
IF EOF(−1) THEN 480.

INPUT $−1
Introduce datos del cassette. El fichero debe estar abierto.
INPUT # −1, A, B, C.

MOTOR
Enciende y apaga el motor del cassette.
MOTOR ON.
MOTOR OFF.

OPEN a, $−1, *nombre*
Abre el canal con el cassette, «*a*» puede ser «1» (entrada) o «0» (salida). El nombre es el de un fichero de datos.
OPEN "I", # −1, X9, LN(J).

PRINT # −1
Escribe datos en el cassette. El fichero debe estar abierto.
PRINT # −1, X9, LN(J).

SKIPF
Salta hasta el final del programa especificado.
SKIPF "NOMBRE".

Sentencias de control de impresión

LLIST
Saca por impresora las líneas de programa especificadas.
LLIST.
LLIST 10-95.

OPEN "0", # −2, nombre de fichero
Abre el canal de salida hacia la impresora.
OPEN "0", # −2, "SALIDA".

PRINT # −2
Imprime la lista de elementos por la impresora.
PRINT # −2, W, A$, X.

Ordenes del sistema

CONT
Continúa la ejecución de un programa después de una sentencia STOP o de BREAK.

DEL
Borra líneas de programa.
DEL 100-350.
DEL 10-
DEL -80.

EDIT
Sirve para alterar el contenido de la línea especificada.
EDIT 115 [ENTER].
Una vez en el modo de edición se podrán usar los subcomandos:
*n*C Cambia *n* caracteres.
*n*D Borra *n* caracteres.
I Inserta caracteres.

H Borra el resto de la línea e inserta caracteres.
L Lista el estado actual de la línea.
*n*S*c* Busca la *n*-sima ocurrencia de *c*.
X Extiende la línea.
n<- Mueve el cursor *n* espacios a la izquierda.
[SHIFT][^] Sale del subcomando.
n[espacio] Mueve el cursor *n* espacios a la derecha.

LIST
Presenta líneas por pantalla.
LIST.
LIST 10-95.
LIST -200.

NEW
Borra el programa de memoria.

RENUM *nueva línea, línea de comienzo, incremento.*
Permite renumerar parte o todas las líneas de programa.
RENUM.
RENUM 100, 50, 10.
RENUM,, 20.

RUN
Ejecuta un programa comenzando por la línea menor o la dada.
RUN.
RUN 250.

TROFF
Desactiva el seguimiento del flujo del programa.
TROFF.

TRON
Activa el seguimiento.

Variables simples

Tipo	*Nombre*	*Rango*
Numérico Cadena	AB AB$	+-10E38 0 a 255 caracteres

Donde A deberá ser una letra, B es opcional y puede ser una letra o dígito. Sólo los dos primeros caracteres de un nombre son significativos.

Matrices

Tipo	*Nombre de un elemento*
Numérica Cadena	AB(3, 4, 5) AB$(17, 8, 2)

Los nombres y rangos funcionan igual que para variables simples. La dimensión de una matriz está limitada por la memoria.

Caracteres especiales

'	Abreviatura de REM.
?	Abreviatura de PRINT.
:	Separa sentencias en una línea.
$	Al final de una variable, la hace de tipo cadena.

Códigos de error

/0	División por cero.
AO	Fichero ya abierto.
BS	Indice erróneo.
CN	No se puede continuar.
DD	Intento de redimensionar una matriz.
DS	Sentencia directa en un fichero.
FC	Llamada ilegal de función.
FD	Datos erróneos.
FM	Modo erroneo de fichero.
ID	Sentencia directa errónea.
IE	Entrada pasado el final del fichero.
I/O	Error de entrada/salida.
LS	Cadena demasiado larga.
NF	NEXT sin FOR.
NO	Fichero sin abrir.
OD	Sin datos en READ.
OM	Sin memoria.
OS	Sin espacio para cadenas.
OV	Desbordamiento, número muy grande.
RG	RETURN sin GOSUB.
SN	Error de sintaxis.
ST	Fórmula de cadena demasiado compleja.
TM	Los tipos no casan.
UL	Línea sin definir.

Sentencias para gráficos

CIRCLE(x, y)r, c, h, s, e
Dibuja un círculo con centro en (x, y), radio r, y color c. «h» es la relación alto/ancho. (El círculo puede empezar en s y acabar en e).
CIRCLE(128, 96), 25.
CIRCLE(100, 50), 30, 4, 1, 0, 0.5.

COLOR primer término, fondo
Fija los colores utilizados como fondo y en primer término, de entre los colores posibles.
COLOR 0, 5.

DRAW cadena
Dibuja líneas siguiendo las instrucciones de la cadena:
Mn, x, y Muévete a x, y.
Un, n puntos hacia arriba.
Dn, n puntos hacia abajo.
Ln, n puntos a la izquierda.
Rn, n puntos a la derecha.
En, n puntos a 45 grados.
Fn, n puntos a 135 grados.
Gn, n puntos a 225 grados.
Hn, n puntos a 315 grados.
Ak, Gira un ángulo según k.
0=0 grados, 1=90 grados, 2=180 grados, 3=270 grados.
Sk, Dibujo a escala por el factor k/4 (k entre 1 y 62).
N, sin actualizar la posición de dibujo.
B, Movimiento en blanco.
C, Fija el color de la línea.
X, Ejecuta una subcadena.

GET(x1, y1)−(x2, y2), matriz, 6
Lee el gráfico contenido en el área de pantalla definida por el rectángulo cuyo ángulo superior izquierdo es x1, y1 y el inferior derecho es x2, y2. G es opcional y recoge los gráficos con todo detalle.
GET(0, 0)−(24, 24), X, G.

LINE(x1, y1)−(x2, y2), a, b
Dibuja una línea desde (x1, y1) hasta (x2, y2). Si se omite (x1, y1) se tomará el último punto utilizado. «a» deberá ser PSET (coge el color en primer término), o PRESET (coge el color de fondo). «b» es opcional, y deberá ser B (dibuja un rectángulo de esquinas (x1, y1) y (x2, y2)) o BF (rellena el rectángulo con el color utilizado para los primeros términos).
LINE(10, 20)−(35, 15), PSET.
LINE(0, 0)−(128, 96), PSET, BF.

PAINT(x, y), c, b
Pinta los dibujos en pantalla con el color c, a partir del punto (x, y) hasta el borde de color b.
PAINT(128, 96), 1, 4.

PCLEAR n
Reserva n páginas de memoria para gráficos (n< =8).
PCLEAR 6.

PCLS c
Limpia la pantalla en alta resolución con el color c. Vea CLS para los códigos de color.
PCLS 5.

PCOPY a TO b
Copia la página a de gráficos en la página destino b.
PCOPY 3 TO 4.

PMODE modo, página
Selecciona el modo de resolución (de 0 a 4) y la primera página de memoria (de 1 a 8).
PMODE 4, 1.

PRESET(x, y)
Desactiva el punto (x, y), dándole el color de fondo.
PRESET(15, 37).

PSET(x, y, c)
Activa el punto (x, y) con el color c.
PSET(120, 95, 4).

PUT(x1, y1)–(x2, y2), matriz, acción
Pone el gráfico de la matriz en el rectángulo en pantalla definido por (x1, y1) y (x2, y2).
La acción es opcional, y puede ser PSET, PRESET, AND, OR o NOT.
PUT(A, C)–(A+24, C+24), M.

RESET(x, y)
Cambia el color del punto (x, y) en baja resolución, al color de fondo.
RESET (10, 12).

SCREEN tipo, juego
Selecciona el tipo de pantalla (texto=0, gráficos=1) y el juego de colores (0 ó 1).
SCREEN 1, 1.

SET(x, y, c)
Fija el punto (x, y), en baja resolución, con el color c.
SET(15, 30, 8).

Funciones de cadena

Argumentos de las funciones: N, M representan valores numéricos enteros; X$, Y$ son cadenas.

ASC(X$)
Devuelve el código ASCII del primer carácter de la cadena.
X=ASC(A$).

CHR$(N)
Devuelve el carácter correspondiente al código ASCII especificado.
A$=CHR$(143).

HEX$(N)
Devuelve el valor hexadecimal del argumento como una cadena.
PRINT HEX$(46).

INKEY$
Muestra el teclado y devuelve el carácter pulsado.
K$=INKEY$.

INSTR(N, X$, Y$)
Busca la primera ocurrencia de la cadena Y$ a partir de la posición N de la cadena X$.
P=INSTR(B, A$, «TED»).

LEFT$(X$, N)
Devuelve los primeros N caracteres de la cadena X$.
B$=LEFT$(A$, 9).

LEN(X$)
Devuelve la longitud de X$.
K=LEN(K$).

MID$(X$, N, M)
Devuelve una subcadena X$, de longitud M, comenzando en la posición N.
B$=MID$(A$, 4, 1).

RIGHT$(X$, N)
Devuelve los últimos N caracteres de la cadena X$.
B$=RIGHT$(A$, 3).

STRING$(N, X$)
Devuelve una cadena formada por N copias del primer carácter de X$. El código ASCII del carácter puede reemplazar a X$.
L$=STRING$(32, «+»).
PRINT STRING$(5, 41).

STR$(N)
Convierte N a su representación como cadena.
X$=STR$(14, 4).

VAL(X$)
Convierte los caracteres numéricos de X$ a números.
P=VAL(D$).

Funciones numéricas

Argumentos:	a	valor numérico 10E38.
	n	valor entero 0-65535.
	r	ángulo expresado en radianes.
	x, y	posición en baja resolución x: 0-63 y: 0-31.
	w, z	posición en alta resolución w: 0-255 y: 0-191.

ABS(a)
Calcula el valor absoluto.
AY=ABS(Y).

ATN(a)
Devuelve el arcotangente en radianes.
X=ATN(A).

COS(a)
Devuelve el coseno del ángulo dado en radianes.
W=COS(2*B).

EXP(a)
Devuelve la exponencial de un número, e^a.
Q=EXP(−5.6).

FIX(a)
Devuelve el valor truncado.
Z=FIX(13.43).

INT(a)
Convierte un número en entero.
A=INT(5.7).

JOYSTK(a)
Devuelve las coordenadas horizontal o vertical de los joysticks.
0 = horizontal, joystick izq.
1 = vertical, joystick izq.
2 = horizontal, joystick der.
3 = vertical, joystick der.
H=JOYSTK(0).
V=JOYSTK(P).

LOG(a)
Devuelve el logaritmo natural.
Z=LOG(7.842).

MEM
Devuelve la cantidad de memoria desocupada.
PRINT MEM.

PEEK(n)
Devuelve el contenido de la posición n de memoria.
P=PEEK(65082).

POINT(x, y)
Muestrea la celda para gráficos (x, y). Devuelve el código de color si está activa, 0 si no lo está, o −1 si es un carácter.
T=POINT(15,25).

POS(d)
Devuelve la posición actual de impresión del dispositivo d.
P=POS(−2).

PPOINT(w, z)
Muestrea la celda (w, z) en alta resolución. Devuelve el código de color si está activa ó 0 si no lo está.
P=PPOINT(186, 54).

RND(n)
Genera un número entero aleatorio entre 1 y n. Si n es 0 devuelve un real entre 0 y 1.

SGN(a)
Devuelve el signo del número como 1 si es positivo, 0 si es cero ó −1 si es negativo.
G=SGN(4*H/3).

SQR(a)
Devuelve la raíz cuadrada del número.
X=SQR(A+7*C).

TAN(r)
Devuelve la tangente del ángulo dado en radianes.
Z=TAN(2*A3).

TIMER
Devuelve el valor actual del timer, o permite ajustarlo.
T=TIMER.
TIMER=0.

USR(n)
Llama a una rutina de usuario en lenguaje máquina.
F=USR(D).

VAPTR(var)
Devuelve la dirección de la posición de la variable.
Z=USR(VARPTR(F)).

Indice de materias

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

OTRAS OBRAS DE INTERES PUBLICADAS POR McGRAW-HILL

DICCIONARIOS PUBLICADOS POR McGRAW-HILL

ISBN: 968-422-019-7




Programación BASIC
para microcomputadoras
L. Joyanes Aguilar
Mc Graw Hill